Universidade de Aveiro
2015

Departamento de Eletrónica,
Telecomunicações e Informática

**Diogo Andril
da Silva Branco**

Reserva de Recursos MSRP para o Switch de Tempo-Real HaRTES

MSRP Resource Reservation for HaRTES Real-Time Switch


Universidade de Aveiro
Departamento de Eletrónica,
Telecomunicações e Informática
2015

Diogo Andril da Silva Branco

Reserva de Recursos MSRP para o Switch de Tempo-Real HaRTES

MSRP Resource Reservation for HaRTES Real-Time Switch

"*Normal people... believe that if it ain't broke, don't fix it. Engineers believe that if it ain't broke, it doesn't have enough features yet.*"

— Scott Adams

Universidade de Aveiro

Ano 2015

Departamento de Eletrónica,

Telecomunicações e Informática

Diogo Andril da Silva Branco

Reserva de Recursos MSRP para o Switch de Tempo-Real HaRTES

MSRP Resources Reservation for HaRTES Real-Time Switch

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Eletrónica e Telecomunicações, realizada sob a orientação científica do Prof. Doutor Paulo Bacelar Reis Pedreiras, Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, e do Prof. Doutor Pedro Alexandre Sousa Gonçalves, Professor Adjunto da Escola Superior de Tecnologia e Gestão de Águeda.

Este trabalho é financiado por Fundos Nacionais através da FCT – Fundação para a Ciência e a Tecnologia no âmbito do projeto Serv-CPS -PTDC/EEA-AUT/122362/2010

FCT Fundação para a Ciência e a Tecnologia

MINISTÉRIO DA EDUCAÇÃO E CIÊNCIA

Dedico este trabalho à minha família pelo incondicional apoio e paciência, e em especial à minha namorada pelo seu incansável amor que a torna no pilar da minha vida.

o júri

presidente

Prof. Doutor Rui Manuel Escadas Martins
Professor Auxiliar da Universidade de Aveiro

arguente

Prof. Doutor Luís Miguel Pinho de Almeida
Professor Associado da Universidade do Porto – Faculdade de Engenharia

orientador

Prof. Doutor Pedro Alexandre Sousa Gonçalves
Professor Adjunto da Escola Superior de Tecnologia e Gestão de Águeda

agradecimentos

Quero deixar o meu agradecimento a todas as pessoas que diretamente ou indiretamente contribuíram para a elaboração deste trabalho.

Gostaria de agradecer ao meu orientador, Professor Doutor Paulo Bacelar Reis Pedreiras, pelo apoio prestado durante o desenvolvimento do trabalho realizado, tendo excedido generosamente qualquer expectativa relativa à sua função de orientador. Agradeço também a prontidão e disponibilidade que demonstrou na resolução de qualquer contratempo. Pela amizade, confiança e apoio demonstrado o meu sincero e profundo agradecimento.

Gostaria de agradecer ao meu coorientador, Professor Doutor Pedro Alexandre Sousa Gonçalves, por ter facilitado o meu trabalho com o seu apoio constante ao longo desta etapa. Agradeço também pela sua compreensão perante qualquer contratempo, demonstrada através de sugestões ou estímulos essenciais para o desenvolvimento do trabalho realizado. Pela amizade, confiança e apoio demonstrado o meu sincero e profundo agradecimento.

Gostaria de agradecer ao grupo de Sistemas Embutidos, no seio do qual fui acolhido e onde desenvolvi o meu trabalho. Agradeço a todos pelas discussões, sugestões e as amizades desenvolvidas ao longo deste período.

Gostaria de agradecer à minha família e em especial à minha namorada, pelo apoio incondicional e paciência que demonstraram nos bons e maus momentos do meu percurso académico.

Agradeço também ao Instituto de Telecomunicações (IT), pólo de Aveiro, por ter disponibilizado os recursos necessários para a realização desta dissertação.

A todos um sincero e profundo obrigado.

palavras-chave

Sistemas de Tempo-Real, *Switch* HaRTES, Reserva de Recursos, MSRP, Comunicações de Tempo-Real, Qualidade de Serviço, Ethernet.

resumo

Os mecanismos e técnicas do domínio de Tempo-Real são utilizados quando existe a necessidade de um sistema, seja este um sistema embutido ou de grandes dimensões, possuir determinadas características que assegurem a qualidade de serviço do sistema. Os Sistemas de Tempo-Real definem-se assim como sistemas que possuem restrições temporais rigorosas, que necessitam de apresentar altos níveis de fiabilidade de forma a garantir em todas as instâncias o funcionamento atempado do sistema.

Devido à crescente complexidade dos sistemas embutidos, empregam-se frequentemente arquiteturas distribuídas, onde cada módulo é normalmente responsável por uma única função. Nestes casos existe a necessidade de haver um meio de comunicação entre estes, de forma a poderem comunicar entre si e cumprir a funcionalidade desejadas. Devido à sua elevada capacidade e baixo custo a tecnologia *Ethernet* tem vindo a ser alvo de estudo, com o objetivo de a tornar num meio de comunicação com a qualidade de serviço característica dos sistemas de tempo-real.

Como resposta a esta necessidade surgiu na Universidade de Aveiro, o *Switch* HaRTES, o qual possui a capacidade de gerir os seus recursos dinamicamente, de modo a fornecer à rede onde é aplicado garantias de Tempo-Real. No entanto, para uma arquitetura de rede ser capaz de fornecer aos seus nós garantias de qualidade serviço, é necessário que exista uma especificação do fluxo, um correto encaminhamento de tráfego, reserva de recursos, controlo de admissão e um escalonamento de pacotes. Infelizmente, o *Switch* HaRTES apesar de possuir todas estas características, não suporta protocolos standards.

Neste documento é apresentado então o trabalho que foi desenvolvido para a integração do protocolo SRP no Switch HaRTES.

keywords Real-Time Systems, Switch HaRTES, Resource Reservations, MSRP, Real-Time Communications, Quality of Service, Ethernet.

abstract

The tools and mechanisms of the Real-Time domain are used when there is a need for a system, either an embedded or a large scale system, to possess certain characteristics that ensure the quality of service. The Real-Time Systems are defined as systems that possess strict time constraints, they require high levels of reliability as a way to ensure the timely operation in every instance of the system.

With the growing complexity of the embedded systems, distributed architectures are normally employed, where each module is usually responsible for a single function. In these cases there is a need for a means of communication between them, so that they can communicate with each other and fulfill the desired functionality. Due to its high capacity and low-cost, Ethernet technology has been the target of study, aiming to make a means of communicating with the service quality feature of real-time systems.

In response to this need, arose at the University of Aveiro the Switch HaRTES, which has the ability to manage their resources dynamically in order to provide the network where it is applied Real-time guarantees. However, for a network architecture to be able to provide such guarantees, it must be able to specify the flow of data, do a correct traffic routing, reserve resources, and do an admission control and packet scheduling. Unfortunately, the HaRTES Switch, despite having all these features, does not support a standard protocol.

In this document is then presented the work that was developed for the integration of the SRP protocol in the HaRTES Switch.

Índice

1. Introdução

A rapidez e capacidade do ser Humano crescer, evoluir e adaptar-se é uma das características que mais o define. O ser Humano possui a habilidade de transformar radicalmente o mundo à sua volta. Tal habilidade permite-lhe ter ao seu dispor tecnologia extraordinariamente avançada comparativamente há 20 anos atrás. Esta tornou-se de tal forma predominante, que no mundo civilizado não existe provavelmente ninguém que ainda não tenha ouvido falar de um computador. Com a introdução dos computadores, e de todo o domínio digital, o ser Humano conseguiu melhorar os seus mecanismos de comunicação, e criar uma rede que se estende por todo o mundo, que nos une como nunca antes. Este avanço tecnológico tem vindo a acontecer também na sua capacidade computacional, havendo equipamentos cada vez mais poderosos e mais práticos. Por se tornarem tão práticos para a nossa sociedade, é impossível deixar de reparar o quão presente eles estão no nosso dia-a-dia. Estes vão desde sistemas computacionais super poderosos, que controlam os sistemas mais críticos, até aos sistemas computacionais mais limitados, concebidos apenas para realizar uma única função, eventualmente muito simples c.e.g. escova de dentes eletrónica. Vivemos assim numa era onde nos encontramos rodeados de sistemas computacionais, sem que muitas vezes nos apercebamos disso, os denominados sistemas embutidos. No entanto existe no ser Humano uma grande vontade de ser um com o seu meio, e facilitar a sua integração e interação com este, e é possível ver exatamente esta dinâmica nos sistemas computacionais. Agrupando certos

conjuntos de sistemas embutidos, através de meios de comunicação entre estes, para os tornar mais autossuficientes, com o intuito de realizar ações cada vez mais complexas.

Apesar deste tipo de sistemas estar presente em todos os géneros de áreas, existem algumas onde apenas ser autossuficiente ou consciente do meio em que se encontra simplesmente não é satisfatório. Sistemas onde a segurança de pessoas e bens possam estar em risco, necessitam de mais do que uma perspetiva do melhor processamento possível ou da melhor velocidade de comunicação possível. É preciso garantir que tanto ao nível de processamento de eventos, como ao nível da comunicação entre dispositivos, estes sistemas irão realizar as suas ações atempadamente. Estes sistemas apresentam restrições temporais, que têm que ser garantidas em todas as condições de funcionamento, denominando-se Sistemas de Tempo-Real. Estes estão presentes nas áreas mais importantes e críticas da nossa sociedade.

Os sistemas tempo-real têm sofrido uma grande evolução ao longo do tempo, quer na vertente computacional que na infraestrutura de comunicações. Neste último caso, tem havido uma forte aposta na adoção de protocolos de comunicação usados em redes de dados genéricas (Ethernet, IEEE802.11, etc.), por contraponto aos protocolos específicos tradicionalmente usados em sistemas embutidos distribuídos (vulgarmente denominados *fieldbuses*). No que concerne a redes cabladas, o protocolo Ethernet tem merecido um especial interesse, fruto da sua elevada largura de banda, baixo custo, elevada disponibilidade e uso massivo. Todavia, este protocolo não suporta nativamente comunicações tempo-real, pelo que ao longo do tempo foram desenvolvidas diversas técnicas para obter este tipo de serviço. Na Universidade de Aveiro tem vindo a desenvolver-se um *switch* de *Ethernet* com capacidades especiais que assenta o seu funcionamento ao nível das comunicações no paradigma *Flexible Time Triggered* (FTT), denominado *switch* HaRTES. Este é capaz de fornecer aos dispositivos a si conectados garantias de

recursos na rede, para que as suas comunicações se realizarem atempadamente. Para que estas garantias possam ter utilidade prática, é necessário que existam mecanismos de interface que permitam às aplicações especificar os seus requisitos, consubstanciados por meio de reservas. Adicionalmente, é também essencial que estes mecanismos sejam baseados em *standards*, possibilitando assim o posicionamento deste *switch* para utilização fora do meio académico.

Assim sendo, tem vindo a ser objeto de desenvolvimento, a adaptação do *switch* HaRTES para que este suporte normas *standard* do mercado da indústria automóvel. No presente trabalho é apresentado o desenvolvimento do mecanismo de reserva de recursos denominado *Stream Reservation Protocol* (SRP). No âmbito desta dissertação serão apresentadas aplicações, arquiteturas e blocos funcionais que irão possibilitar que o *switch* HaRTES seja totalmente compatível com SRP.

1.1 Organização do Documento

Este documento encontra-se dividido em sete capítulos, organizado como se segue:

- **Capítulo 1 Introdução**: contextualização do cenário que deu motivação ao desenvolvimento da dissertação e os seus principais objetivos;

- **Capítulo 2 Conceitos Essenciais de Tempo-Real:** introdução dos conceitos de Sistemas de Tempo-Real necessários para a compreensão das necessidades e mecanismos associados ao desenvolvimento da dissertação;

- **Capítulo 3 Conceitos Essenciais de Comunicações em Tempo-Real:** introdução aos mecanismos utilizados nas comunicações *Ethernet* e técnicas desenvolvidas para dotar as redes de *Ethernet* de comunicações com garantias de qualidade de serviço.

- **Capítulo 4 SRP – Stream Reservation Protocol:** contextualização das necessidades, características e mecanismos decretadas pela norma, com enfase no registo de reservas na rede;

- **Capítulo 5 Implementação:** contextualização do estado do projeto de fornecer o *switch* HaRTES com capacidades de reservas de recursos segundo a norma, seguido de uma listagem de todos as aplicações, estruturas e alterações necessárias realizadas durante a dissertação;

- **Capítulo 6 Testes e Validação:** listagem de todos os testes realizados para validação do sistema, assim como os resultados obtidos e sua discussão;

- **Capítulo 7 Conclusão**: conclusões finais sobre o trabalho desenvolvido durante esta dissertação;

2. Conceitos Essenciais de Tempo-Real

2.1 Definição de Sistemas de Tempo-Real

A nossa sociedade vive neste momento num mundo rodeado de sistemas computacionais, desde os sempre presentes *smartphones* e televisões inteligentes, aos automóveis e sistemas de segurança. Cada um mais complexo do que o anterior, mas sempre concebidos com uma forte interação com o utilizador ou com o meio envolvente. É possível também verificar que qualquer falha no sistema de uma televisão inteligente não pode ser equiparado às consequências que uma falha no sistema de segurança de uma fábrica de produtos perigosos poderia ter. Sendo assim, existem vários tipos de sistemas computacionais, cada um muito particular nas suas características. Quando uma destas características implica o acompanhamento de acontecimentos no seu meio envolvente e atuação atempada sobre este, entra-se no domínio de Sistema de Tempo-Real.

Existem inúmeras definições de Sistema de Tempo-Real. No entanto todas elas dão grande relevância à perspetiva temporal num sistema computacional. Desta forma, quando se tenta definir este conceito, é imprescindível a noção de tempo de resposta e de *deadline*. O tempo de resposta pode ser definido como o tempo que um sistema demora a gerar um *output* após um determinado *input*, e a *deadline* como o limite temporal para um determinado acontecimento. Com estas noções podemos então tentar definir o conceito de Sistema de Tempo-Real, como um sistema que necessita de processar informação e produzir uma resposta dentro

de um determinado período de tempo. Caso isto não se cumpra, pode ter consequências, eventualmente catastróficas.

Os Sistemas de Tempo-Real têm tipicamente que obedecer a vários requisitos. Entre estes podem evidenciar-se três tipos: os funcionais, os temporais e os de dependabilidade. A nível funcional, como já foi mencionado, este tipo de sistema necessita de acompanhar o seu meio envolvente e realizar uma atuação sobre este. Para isto, muito à semelhança dos sistemas de controlo, este necessita de recolher informação sobre o meio através de sensores, ser capaz de processar os dados amostrados por estes através de um sistema controlador, e por fim ser capaz de atuar sobre o meio com o uso de atuadores. Esta estrutura pode ser visualizada na figura 2.1.

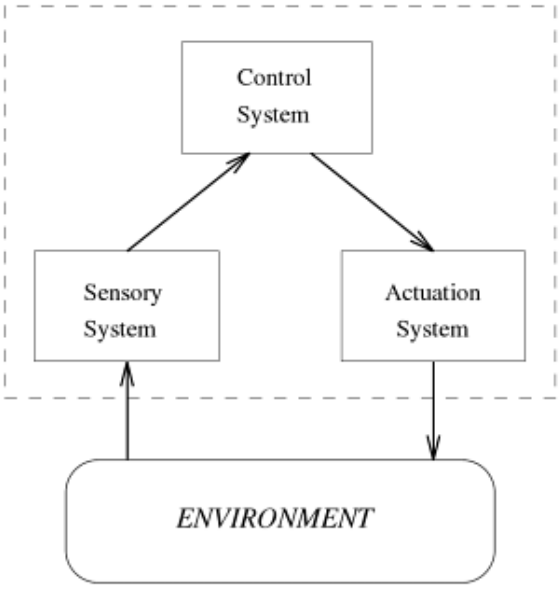

Figura 2.1: Diagrama de blocos de um sistema de controlo de tempo-real [1]

Os requisitos temporais, que derivam diretamente do meio ambiente que o sistema acompanha, impõem diversas restrições, tais como os atrasos de amostragem e processamento e as variações destes (*jitter*). Desta forma, todos os tempos e atrasos terão de ser contabilizados e todas as instâncias têm que obedecer às restrições temporais. De acordo com a utilidade da informação para a

aplicação em questão, é possível atribuir diferentes classificações às restrições temporais. *Butazzo*[1] classifica estas como:

- **Soft**, quando há uma degradação da utilidade da informação ao longo do tempo, após ultrapassado a *deadline*;

- **Firm**, quando a utilidade da informação se perde completamente após ultrapassado a *deadline*;

- **Hard**, quando a violação da *deadline* pode originar falhas catastróficas.

Recorrendo a estas classificações é possível também classificar os Sistemas de Tempo-Real [2]. Estes são classificados como:

- **Soft Real-Time**, quando as restrições temporais no sistema apenas se apresentam como sendo do tipo *firm* ou *soft*.

- **Hard Real-Time**, quando dentro do conjunto de restrições temporais do sistemas existe pelo menos uma restrição do tipo *hard*.

Devido à importância das restrições temporais do tipo *hard*, e a estas estarem associadas a aplicações críticas com graves consequências no caso de falha, na conceção de Sistemas de Tempo-Real existem elevados requisitos de dependabilidade. Desta forma, é necessário que as aplicações desenvolvidas apresentem altos níveis de fiabilidade, garantindo em todas as instâncias o correto funcionamento do sistema através do estudo dos cenários de pior caso, em particular do tempo de resposta. Construindo um sistema obedecendo a estes requisitos, é possível criar um Sistema de Tempo-Real com garantias e qualidade de serviço (*QoS*), pois este irá apresentar, mesmo face ao pior caso possível, recursos suficientes para o seu correto funcionamento [2].

2.2 Características dos Sistemas Operativos de Tempo-Real

Na figura 2.3 encontra-se retratado o modelo genérico dos sistemas operativos de tempo-real e as suas principais características. É possível verificar a existência de tarefas, gestor de recursos, escalonador, *dispatcher* de tarefas, e um gestor de tempo.

As tarefas num Sistema de Tempo-Real são consideradas como a sua componente elementar, na qual as restantes componentes irão realizar operações sobre estas ou apresentar algum tipo de interação com elas. De uma forma sucinta, no modelo mencionado, as tarefas apresentam-se como um conjunto de instruções que o processador necessitará de executar, podendo estas interagir com os recursos disponíveis ou com o meio envolvente.

O gestor de tarefas estará encarregue da criação, destruição e atualização do estado em que se encontram as tarefas, seguindo o modelo presente na figura 2.2. Dependendo dos casos, existe um conjunto de estados que uma tarefa pode tomar, estes são: *idle*, *ready*, *executing*, *blocked* e *suspended*.

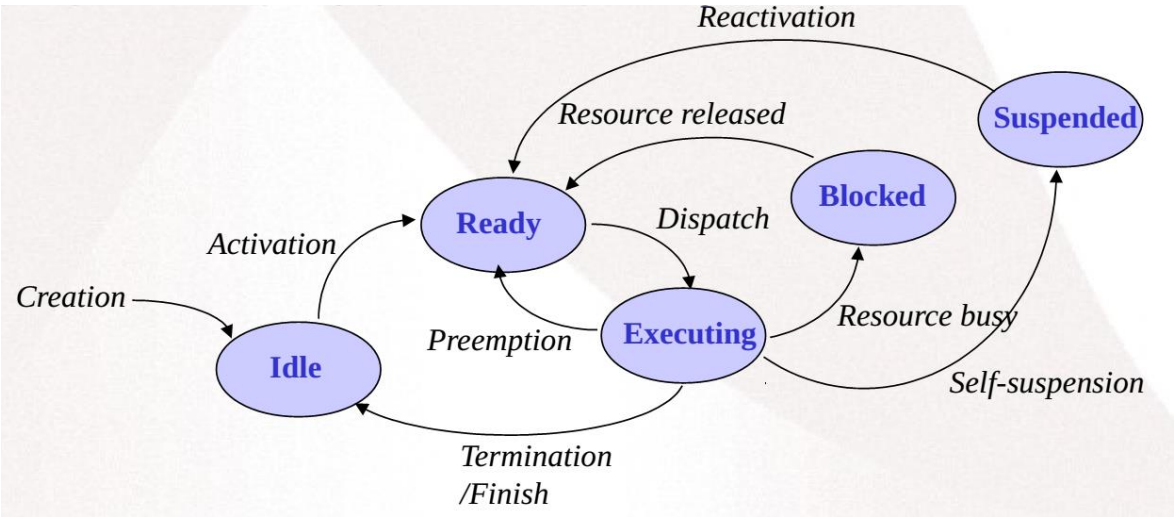

Figura 2.2: Diagrama de estados de uma tarefa num Sistema de Tempo-Real [3]

O escalonador tem como função ordenar as tarefas segundo um critério especificado *a priori*, para posterior execução destas. O escalonador poderá obedecer a vários algoritmos de escalonamento, onde por exemplo a prioridade relativa das tarefas ao longo do tempo é constante (prioridades fixas), ou ao longo do tempo a prioridade relativa destas está sujeita a uma variação (prioridades dinâmicas).

O *dispatcher* está encarregue de colocar em execução a tarefa mais prioritária previamente ordenada pelo escalonador.

O gestor de tempo para além de ser usado para o correto escalonamento das tarefas e realizar verificações temporais, é também usado para a ativação de tarefas que poderão ter um perfil periódico.

Quando existe mais que uma tarefa a requerer o acesso a um recurso que exige acesso exclusivo, existe a necessidade de impedir acessos concorrentes. O gestor de recursos irá então regular a utilização destes consoante um critério específico. No âmbito desta dissertação será necessário esclarecer em que consiste uma tarefa e as suas características específicas, e também explicar em detalhe certas técnicas de escalonamento e gestão de tarefas.

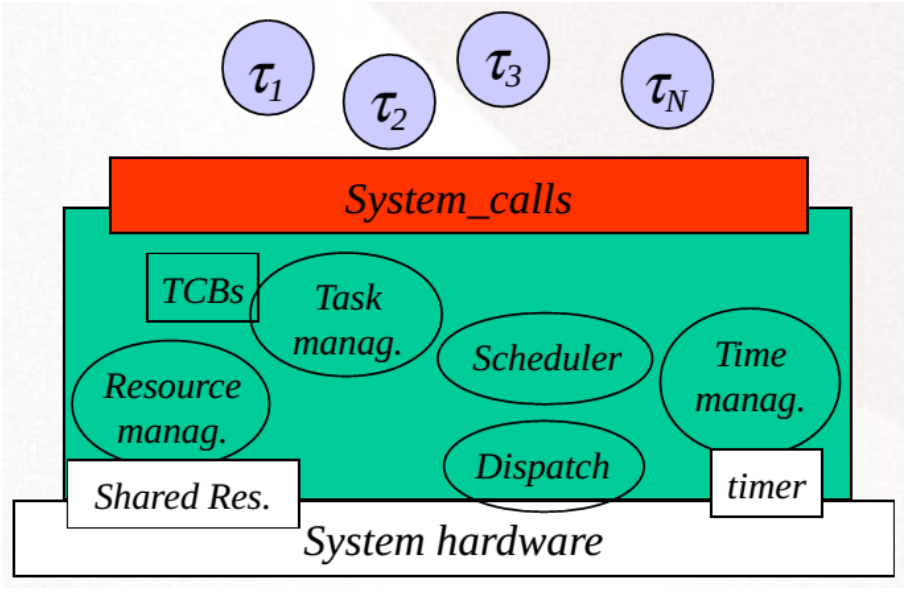

Figura 2.3: Diagrama das componentes características dos Sistemas Operativos de Tempo-Real [3]

2.2.1 Tarefas

Como já foi referido, as tarefas nos Sistemas de Tempo-Real são a componente elementar deste tipo de arquitetura. Nos sistemas computacionais, estas são descritas como um conjunto de instruções que realizam um determinado processamento, sendo que através do tipo de ativação, uma tarefa pode ser classificada como periódica, aperiódica ou esporádica.

Quando uma tarefa é ativada em intervalos de tempo constante, esta tarefa é classificada como tarefa periódica. Caso o intervalo entre ativações não seja constante, mas existe um tempo mínimo entre ativações sucessivas, esta é classificada como uma tarefa esporádica. Em último caso, quando só é possível caracterizar o instante de ativações de forma probabilística, esta é classificada como tarefa aperiódica.

A periocidade de uma tarefa é apenas uma das características intrínsecas a estas. De seguida serão listadas as restantes, segundo *Butazzo* [1], sobre as quais o escalonador irá realizar cálculos para atribuir uma ordem para a execução destas tarefas. Uma tarefa de tempo real τ_i é caracterizada por:

- **Fase relativa Φ_i:** é o instante de tempo onde ocorre a primeira ativação da tarefa;

- **Instante de ativação ou chegada a_i:** instante no qual uma tarefa fica pronta para execução;

- **Tempo máximo de execução C_i:** é o tempo de computação do pior caso de uma tarefa;

- **Período T_i ou *minimal interarrival time* (mit):** no caso das tarefas periódicas é o tempo entre ativações sucessivas da mesma tarefa. No caso das tarefas esporádicas mit é o tempo mínimo entre ativações sucessivas;

- ***Deadline absoluta* d_i:** instante de tempo no qual a tarefa deverá terminar a sua execução;

- ***Deadline relativa* D_i:** diferença temporal entre a ativação de uma tarefa e a sua conclusão;

- ***Start time* s_i:** instante de tempo na qual a tarefa começa a ser executada.

- ***Finishing time* f_i:** instante de tempo na qual a tarefa acaba de ser executada.

- **Tempo residual de computação $c_i(t)$:** medida de tempo de processamento restante num determinado instante *t* até ao final da execução da tarefa.

- **Tempo de resposta R_i:** medida de tempo entre o instante de ativação a_i e o instante f_i quando a tarefa acaba de ser executada.

- ***Lateness* L_i:** medida de tempo entre a *deadline* absoluta d_i e o instante f_i. Este valor será negativo quando a tarefa completa a sua execução antes da *deadline*.

- ***Tardiness* E_i:** tempo durante o qual uma tarefa continua em execução após a sua *deadline*.

- ***Laxity* Xi:** é a medida máxima de tempo que a execução pode ser adiada, sem ser violada a *deadline* absoluta da tarefa.

Na figura 2.4 encontram-se representadas as características essenciais para o âmbito da dissertação.

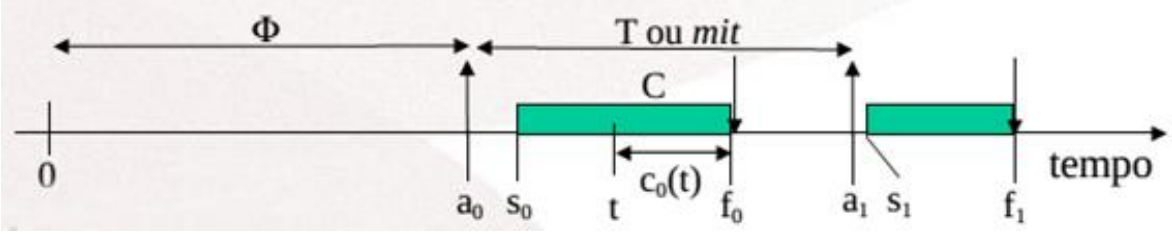

Figura 2.4: Diagrama temporal das características principais de uma tarefa [4]

2.2.2 Técnicas de Escalonamento

Como já mencionado, um escalonador tem como função ordenar um conjunto de tarefas de tempo-real, de modo a que nenhuma destas comprometa a sua *deadline*. Sendo assim, faz sentido que sejam apresentadas alguns dos algoritmos de escalonamento existentes. No entanto, no âmbito desta dissertação, será dado mais relevância aos algoritmos que envolvem o conceito de servidor e o escalonamento de tarefas esporádicas.

Segundo *Butazzo* [1], os de algoritmos de escalonamento de tarefas de tempo-real podem ser classificados segundo as seguintes classes:

- **Preemptivo vs. não-preemptivo**. Quando existe preempção, o algoritmo é capaz de interromper a qualquer instante a execução de uma tarefa, e atribuir o processador a outra tarefa de acordo com uma política de escalonamento. Quando não existe, a execução de uma tarefa não pode ser interrompida até ao fim da sua execução.

- **Estático vs. dinâmico**. Um algoritmo será estático, quando os critérios que são utilizados para o escalonamento das tarefas se mantêm fixos ao longo do tempo. Quando os critérios escolhidos variam ao longo do tempo, este algoritmo será dinâmico.

- **Off-line vs. on-line**. Quando o escalonamento da totalidade de um conjunto tarefas é feito antes de o sistema entrar em funcionamento, este denomina-se *off-line*. Normalmente este escalonamento apresentará um carácter cíclico e será processado pelo *dispatcher*. O escalonamento denomina-se *on-line* quando as decisões são tomadas durante a execução do sistema, nomeadamente nos instantes de ativação e término das tarefas.

- **Ótimo vs. sub-ótimo**. Um algoritmo designa-se ótimo quando encontra garantidamente um escalonamento que garante o cumprimento das *deadlines* de todas as tarefas, caso este exista (dentro da sua classe). Um algoritmo designa-se como sub-ótimo ou heurístico, quando se baseia numa função heurística para as suas decisões, nunca garantido um escalonamento ótimo mas que tende para este.

2.2.2.1 Técnicas de Escalonamento de Tarefas Periódicas

Butazzo [1] refere quatro algoritmos básicos para o escalonamento de tarefas periódicas. Estes englobam escalonamentos de diversas classes, mas maioritariamente regidos pela proximidade da tarefa à sua *deadline*. Estes algoritmos são: Escalonamento estático cíclico, *Rate Monotonic* (RM), *Deadline Monotonic* (DM) e o *Earliest Deadline First* (EDF)

No escalonamento estático cíclico, o escalonamento é construído *a priori* do início do sistema, de acordo com uma divisão temporal fixa por slots/micro-ciclos (μC) todos da mesma duração, onde uma ou mais tarefas são alocadas para execução segundo a sua frequência de ativação de modo a cumprir as suas *deadlines*. Esta distribuição de tarefas, devido a terem um carácter periódico, vai produzir um padrão cíclico global denominado por macro-ciclo (MC). Este macro-ciclo é guardado, definindo a ordem de execução das tarefas durante a execução do sistema. Inicialmente são definidos os micro-ciclos em que as tarefas são ativadas. Após este procedimento são percorridos todos os micro-ciclos e são alocadas as tarefas tendo em conta a capacidade do micro-ciclo e a duração das tarefas. Todas as tarefas que não couberem num específico micro-ciclo são atrasadas para o seguinte onde também será tentada a alocação destas. A seleção das tarefas a atrasar ou a alocar pode seguir um critério específico, como o *Rate Monotonic* ou a *Earliest Deadline First*.

No *Rate Monotonic* a prioridade das tarefas é inversamente proporcional ao seu período. O *Rate Monotonic* tem também uma natureza preemptiva, pois a

chegada de uma tarefa de prioridade superior à que se encontra em execução irá resultar na suspensão desta para executar a de maior prioridade.

O *Deadline Monotonic* em relação ao *Rate Monotonic* difere apenas em que a atribuição de prioridades é definida através da *deadline* relativa das tarefas, sendo que quanto maior a sua *deadline* menor é a prioridade da tarefa. Caso todas as tarefas apresentem *deadline* igual ao período, tanto o RM como o DM produzem o mesmo escalonamento. A escalonabilidade das tarefas pode ser verificada antes do início do sistema através de testes baseados na taxa de utilização do CPU e/ou de tempo de resposta.

O *Earliest Deadline First* é um algoritmo dinâmico que faz o escalonamento das tarefas segundo as suas *deadlines* absolutas. Desta forma, à medida que o instante de tempo se vai aproximando da *deadline* relativa de uma tarefa, esta, se não tiver sido ainda executada, vai aumentando gradualmente a sua prioridade.

2.2.2.2 Técnicas de Escalonamento de Tarefas Aperiódicas e Esporádicas

Como já mencionado anteriormente, a diferença entre uma tarefa periódica e uma esporádica é o seu padrão de ativação. Como as tarefas periódicas apresentam uma frequência de ativação constante, estas possuem os seus instantes de ativação bem instanciados numa linha temporal. Por sua vez, as tarefas esporádicas são caracterizadas por um padrão de ativação variável. No entanto possibilitam delimitar um período temporal onde não irá surgir a próxima ativação. Quando a *deadline* é igual ao período de uma tarefa, o pior caso possível para a ativação das tarefas esporádicas é o mit.

Após esta consideração, *Butazzo* [1] sugere que os algoritmos antes mencionados para as tarefas periódicas, podem também ser aplicados às tarefas esporádicas se for considerado o pior caso possível, ou seja, o período das tarefas esporádicas for igual ao mit.

Em relação às tarefas aperiódicas, devido aos instantes de ativação destas apenas poderem ser previstos através de métodos probabilísticos, é impossível dar garantias de atendimento à chegada das tarefas utilizando os algoritmos anteriormente mencionados.

2.2.2.3 Técnicas de Escalonamento Híbrido

Apesar de terem sido mencionadas técnicas de escalonamento para tipos de tarefas específicas, na verdade, os Sistemas de Tempo-Real mostram-se sempre muito mais complexos, pois raros são os casos onde o conjunto de tarefas do sistema pertencem somente a um tipo. Desta forma, são necessárias técnicas de escalonamento híbridas, capazes de providenciar garantias de atendimento a um conjunto de tarefas de tipos variados.

Butazzo [1] menciona um método simples para lidar com um conjunto tarefas aperiódicas de caracter *soft* mesmo na presença de tarefas periódicas, em que as tarefas aperiódicas são escalonadas em última instância, quando não existe nenhuma tarefa periódica pronta para execução, como exemplificado na figura 2.5. No entanto quando existe uma grande quantidade de ativações de tarefas periódicas, o tempo de resposta para tarefas aperiódicas aumenta e caso estas sejam mais importantes para o sistema do que as periódicas, isto pode fazer com que as deadlines das tarefas aperiódicas não sejam cumpridas.

Apesar disso, é possível melhorar o mecanismo de escalonamento em *background*, através da introdução do conceito de servidor. Segundo *Butazzo* [1], o servidor será uma tarefa periódica ou esporádica que quando atendida irá processar tarefas aperiódicas associadas. À semelhança das tarefas periódicas, o servidor também será caracterizado por um período *Ts* e um tempo de execução, também denominado por capacidade do servidor, *Cs*. Esta capacidade será gasta durante o atendimento das tarefas aperiódicas. Desta forma, as análises de escalonamentos de tarefas periódicas, mencionados anteriormente, podem ser

aplicados a um conjunto híbrido de tarefas, com o objetivo de reduzir o tempo de resposta às tarefas aperiódicas sem prejudicar as garantias do conjunto periódico/esporádico. Convém mencionar que nas figuras que representam o funcionamento do atendimento de tarefas os τ's numerados representam tarefas periódicas no sistema.

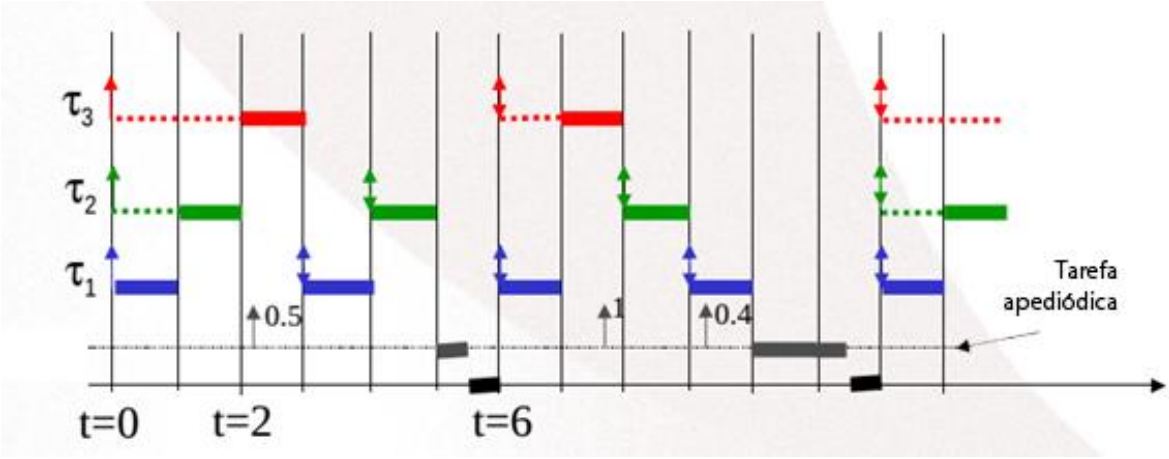

Figura 2.5: Diagrama temporal do funcionamento do atendimento de tarefas aperiódicas em *background* (adaptada de [5])

Há diferentes tipos de servidores, que se distinguem pela forma como a capacidade é gerida. Será de seguida analisado o conceito de três implementações para os servidores de tarefas aperiódicas de prioridades fixas: servidores de sondagem, servidores adiáveis, e os servidores esporádicos.

2.2.2.3.1 Polling Server (Servidor de Sondagem)

No *Polling Server* o servidor é ativado em intervalos regulares, iguais ao seu período *Ts*. Quando ativo, o servidor irá atender as tarefas aperiódicas que eventualmente se encontram em espera dentro dos limites da sua capacidade *Cs*. Caso não exista tarefas aperiódicas pendentes, o servidor suspende até ao instante da sua próxima ativação. Desta forma, a capacidade que seria gasta para o atendimento das tarefas aperiódicas é libertado e o processador encontra-se disponível para atender outras tarefas periódicas. É necessário também explicitar

que, caso chegue um pedido de atendimento para uma tarefa aperiódica após a suspensão do servidor, este pedido só poderá ser atendido na próxima ativação do servidor. A capacidade no caso do *Polling Server* será reposta na totalidade no instante da ativação deste [6,7]. O funcionamento do *Polling Server* encontra-se ilustrado na figura 2.6, onde τ_S representa o servidor que atende tarefas aperiódicas.

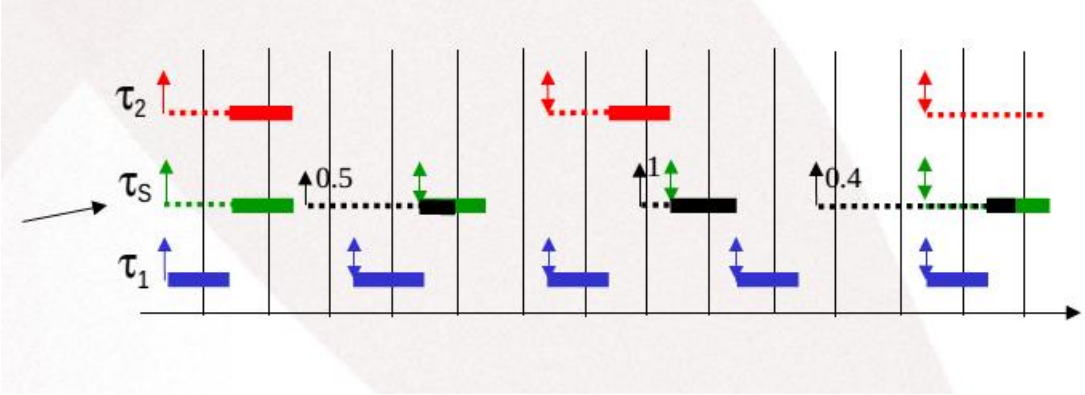

Figura 2.6: Diagrama temporal do funcionamento de um *Polling Server* [5]

2.2.2.3.2 Deferrable Server (Servidor Adiável)

O *Deferrable Server* foi apresentado por *Lehoczky, Sha, e Strosnider* [6,8], com o intuito de melhorar o tempo de resposta aos pedidos das tarefas aperiódicas do *Polling Server*. Em semelhança ao *Polling Server*, este método também mantém o uso do servidor para tarefas aperiódicas como uma tarefa periódica escalonável dentro do conjunto destas. No entanto, a melhoria encontra-se na capacidade de o servidor atender tarefas aperiódicas que possam surgir mesmo depois do instante de ativação, desde que o servidor ainda apresente capacidade suficiente para o realizar. Neste algoritmo a reposição da capacidade do servidor ocorre também no instante da sua ativação, repondo na totalidade a capacidade original do servidor. O funcionamento do *Deferrable Server* encontra-se ilustrado na figura 2.7, onde τ_S representa o servidor que atende tarefas aperiódicas.

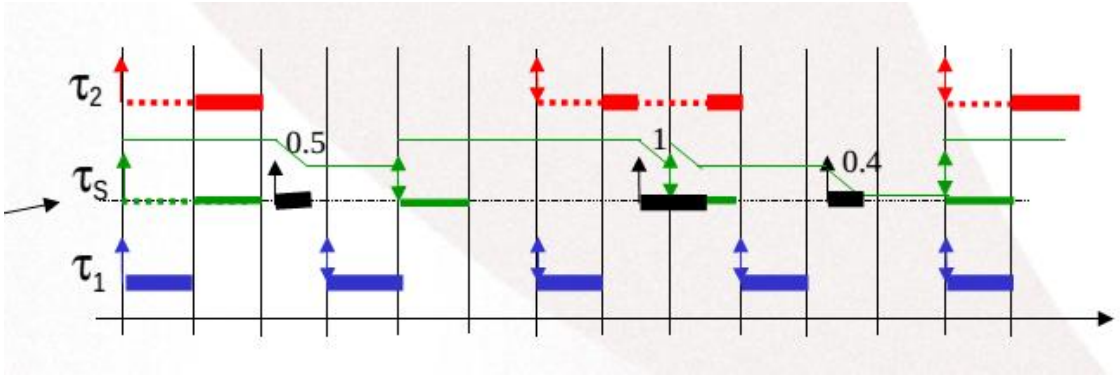

Figura 2.7: Diagrama temporal do funcionamento de um *Deferrable Server* [5]

2.2.2.3.3 Sporadic Server (Servidor Esporádico)

O *Sporadic Server*, é outra proposta por *Lehoczky, Sha, e Strosnider* [7]. O *Deferrable Server* apresenta maior responsividade que o Polling Server, mas introduz uma penalização em termos de escalonabilidade, apresentando uma interferência superior a uma tarefa periódica equivalente (i.e., com o mesmo período e tempo de execução). Este problema encontra-se exemplificado na figura 2.8, onde ocorrem duas instâncias seguidas do servidor devido ao facto de o *Deferrable Server* repor a sua capacidade sem considerar quando é que esta foi consumida. É possível assim concluir que as tarefas de prioridade inferior poderão ter que lidar com uma capacidade instantânea do servidor que poderá neste caso apresentar o dobro da sua capacidade característica. O *Sporadic Server* vem então tentar otimizar os que as duas propostas anteriores contruíram.

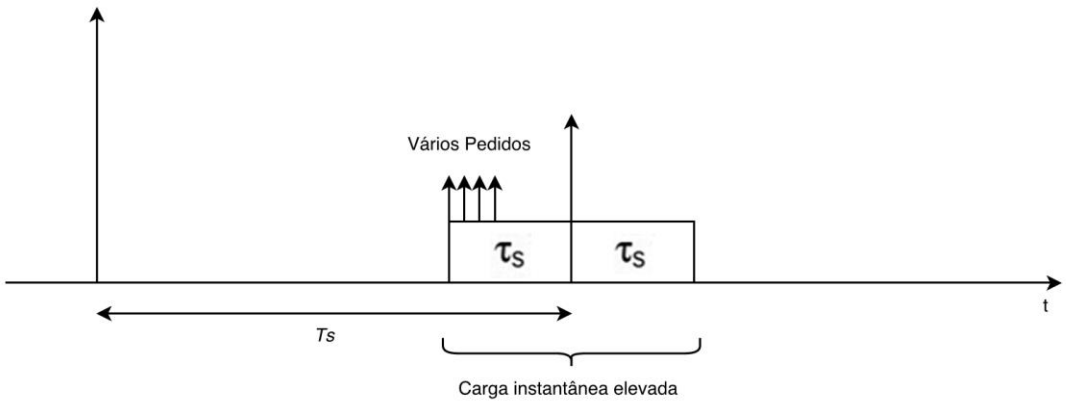

Figura 2.8: Exemplo ilustrativo do problema do *Deferrable Server*

Nos *Sporadic Servers* é criado uma tarefa periódica para o servidor com alta prioridade que irá servir os pedidos de tarefas aperiódicas. Tal como o *Deferrable Server*, este será capaz de preservar a sua capacidade e atender pedidos de tarefas aperiódicas em qualquer altura, desde que a sua capacidade não seja nula. No entanto, não segue o mesmo mecanismo de reposição de capacidade que o *Polling* e o *Deferrable Server*. Esta é apenas reposta um período *Ts* após o atendimento de uma tarefa aperiódica. Desta forma minimiza-se o impacto que um fluxo demasiado grande de tarefas aperiódicas teria nas tarefas periódicas de menor prioridade em relação à tarefa do servidor. O funcionamento do *Sporadic Server* encontra-se ilustrado na figura 2.9.

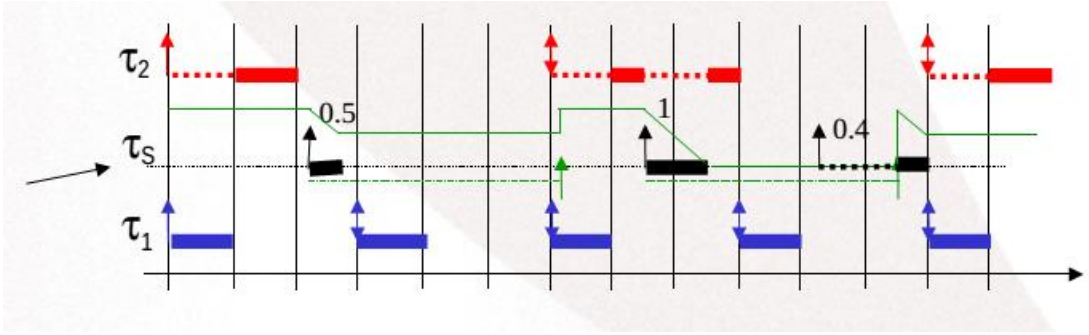

Figura 2.9: Diagrama temporal do funcionamento de um *Sporadic Server* [5]

3. Conceitos Essenciais de Comunicações em Tempo-Real

Como introduzido no início do capítulo dois, muitas das tecnologias atuais assentam na interação com o ser humano ou com ambiente ao seu redor. Estas tecnologias, maioritariamente devido a essa interação, estão implementadas com uma arquitetura de sistemas embutidos distribuídos. Este tipo de arquitetura pode ser encontrada nas mais diversas áreas, no entanto todas as suas aplicações requerem três unidades fundamentais: a unidade sensorial, capaz de fazer medidas ou observações sobre o meio, obtendo assim os dados de *input* do exterior; a unidade de controlo, onde todas as decisões sobre os procedimentos a atuar são calculados e determinados; e a unidade de atuação, onde o sistema é capaz de interagir com o meio segundo as decisões tomadas pela unidade de controlo, produzindo assim um output no exterior [1].

Neste tipo de sistemas, não é obrigatório existir uma só unidade de cada tipo. Na verdade, o que é recorrente neste tipo de sistemas é a utilização de várias unidades sensoriais assim como atuadoras, podendo ou não ter mais que uma unidade de controlo. Este tipo de arquitetura distribuída permite que os sistemas partilhem recursos, estejam abertos à utilização de equipamentos de diferentes vendedores, processem dados concorrentemente, sejam escaláveis e tolerantes a falhas.

Os sistemas com este tipo de arquitetura têm inerente a si o conceito de rede, onde existe entre cada nó um meio de comunicação para troca de dados. No entanto, é importante considerar que para situações em que é necessário prestar qualidade de serviço, como nos Sistemas de Tempo-Real, as trocas de informações na rede apresentam também restrições temporais.

Sendo a *Ethernet* o meio de comunicação mais dominante nas redes cabladas, é natural que se procure desenvolver um protocolo de comunicação que faça uso desta tecnologia. Porém, é necessário assegurar que este tipo de tecnologia providencia garantias de tempo-real mesmo que esta tenha inerente a si um comportamento não determinístico. Ao mesmo tempo, os já existentes protocolos de comunicação para tempo-real baseados em *fieldbuses* e redes de controlo, encontram-se limitados quando comparados à performance que redes baseadas em *Ethernet* apresentam [9]. Segundo *J.D.Decotigni*e [9], dentro destes protocolos, embora antiquados, é possível referir o *WorldFIP* [10], *Profibus* [11], *P Net* [12], *Interbus* [13], *AS-Interface* [14], *SERCOS* [15], *LonWorks* [16], *MVB* [17], *MIL-STD 1553* [18], *DeviceNet* [19], *SDS* [20] e *CAN* [21].

No contexto desta dissertação, será necessário esclarecer como as unidades de um sistema distribuído podem comunicar através da *Ethernet*, dando garantias de serviço a nível da sua pontualidade, através da introdução de alguns conteúdos e protocolos neste momento em desenvolvimento.

3.1 Ethernet

A Ethernet encontra-se como o meio dominante de comunicação nas tecnologias de redes locais ou *LAN's*, estandardizada como IEEE 802.3 [22] que é normalmente conhecido como o protocolo CSMA/CD (*Carrier Sense Multiple Access/Collision Detection*) [24]. Introduzida em 1980, esta foi desenvolvida no *Xerox Palo Alto Research Center* com a parceria das empresas *Intel, Digital Equipment*

Corporation e *Xerox*, podendo tomar velocidades de 10 Mbps, 100 Mbps, 1 Gbps, e até mais recentemente 10 Gbps, através do uso de cabo coaxial, par entrançado e fibra ótica. A comunicação é feita através da troca de pacotes da camada física e da camada de ligação dados do modelo OSI, como declarado no *standard* [22]. O domínio sobre o mercado está diretamente ligado com as seguintes características do protocolo: grande disseminação através dos sistemas informáticos de uso geral; permite implementações de baixo custo; grande flexibilidade na implementação de diferentes topologias; e garante a compatibilidade e utilização dos produtos independentemente dos seus produtores [23].

3.1.1 Ethernet Frame

A tecnologia *Ethernet* consiste no envio de pacotes entre dispositivos compatível com o protocolo IEEE 802.3. Os pacotes que respeitam o protocolo, apresentam todos a mesma estrutura dividida em campos estabelecidos que formam a *Ethernet frame* presente na figura 3.1. Os campos são os seguintes:

- ***Preamble***, mecanismo de sincronização;

- ***Start of Frame***, utilizado para identificar o início da *frame*;

- ***Destination e Source Address***, endereços MAC, representam os identificadores únicos associados aos nós de destino e de origem do pacote;

- ***Ethernet Type***, utilizado para identificar o subprotocolo utilizado;

- ***Data***, utilizado para a transmissão de informação útil, possui um tamanho mínimo de 46 bytes a 1500 bytes;

- ***Frame Check Sequence***, utilizado para a deteção de erros.

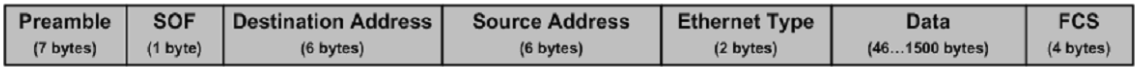

Figura 3.1: Trama de um pacote de dados *Ethernet* II [25]

3.1.2 Switched Ethernet

A Ethernet, como rede composta por ligações físicas, pode assumir topologias diferentes, modificando assim a forma como os nós da rede estão conectados entre si. Na área de redes é possível verificar a existência de diferentes tipos de topologias de ligações, como é o caso das topologias em forma de barramento, anel ou estrela, descritas na figura 3.2. Utilizando a topologia de barramento, os nós irão encontrar-se ligados através de uma linha de comunicação comum que deverá ser partilha por todos. Na topologia em anel, todos os nós da rede encontram-se obrigatoriamente ligados a outros dois nós da rede, formando assim um circuito fechado com ligação direta apenas aos nós conectados a um nó diretamente. Na topologia em estrela, todos os nós de uma rede encontram-se ligados apenas a um nó central, sendo este um *switch* ou um *hub*, que serve de *link* de comunicação com os restantes nós [25].

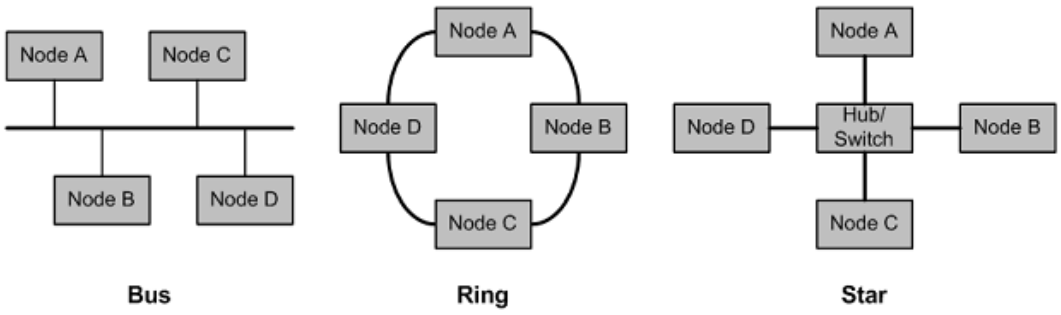

Figura 3.2: Topologias de redes de *Ethernet* [25].

Relevante à causa do fornecimento de qualidade de serviço com a tecnologia *Ethernet*, o mecanismo de CSMA/CD possui um comportamento destrutivo e não determinístico devido à possível colisão de pacotes. No entanto, o uso de um *switch*, e de uma comunicação *full-duplex* numa topologia de estrela, permite criar domínios privados na rede para cada uma das portas do *switch*, onde as colisões de pacotes são totalmente evitadas [26]. Este fator, associado ao facto que este tipo de topologia apresenta débitos elevados de dados, possibilita a

adaptação das redes de *Ethernet* nos Sistemas de Tempo-Real. Estes domínios privados são possíveis pois os *switchs* são capazes de armazenar temporariamente e reencaminhar pacotes por portas específicas, formando assim o conceito de rede *Switched Ethernet*. Isto é feito através de tabelas de reencaminhamento, que podem ser contruídas dinamicamente pelo próprio *switch*, através de diversas técnicas, ou pelo próprio utilizador. Estas tabelas contêm em si informação capaz de associar os endereços MAC dos nós da rede, possíveis de atingir direta ou indiretamente, com a porta do *switch* especifica pela qual isto é possível. Assim sendo, face ao evento da chegada de um pacote por uma das suas portas, o *switch* irá analisar a sua tabela de reencaminhamento. Caso não se verifique a existência de uma entrada na tabela correspondente ao endereço MAC de destino do pacote, o *switch* irá proceder ao *broadcast* do pacote por todas as portas, exceto a porta pela qual este foi recebido. Caso exista uma entrada correspondente, o *switch* irá reencaminhar o pacote pela porta de destino associada a esta entrada.

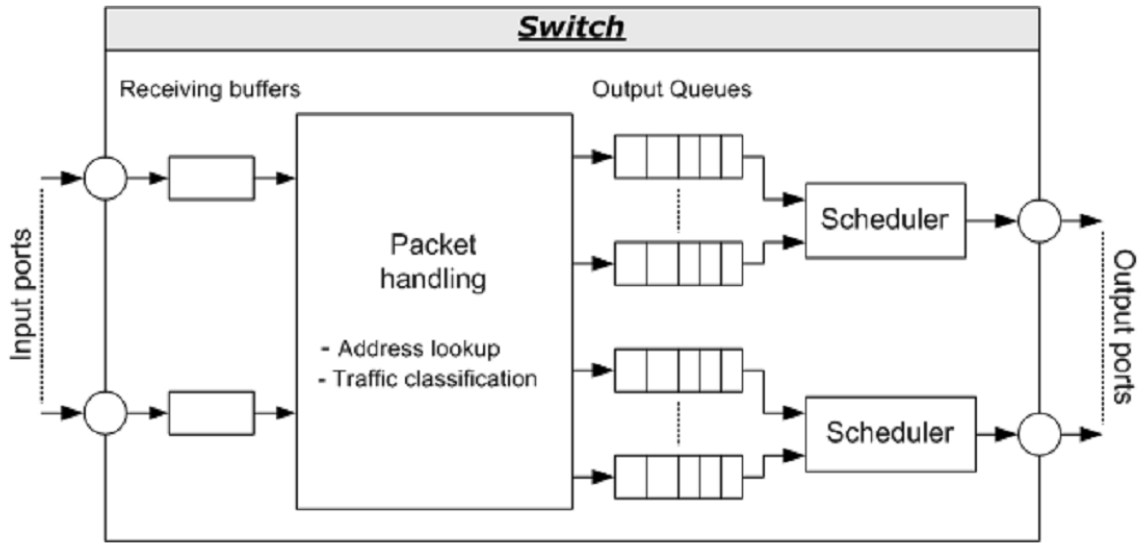

Figura 3.3: Arquitetura genérica de um *switch* [27]

Quando num dado período de tempo chegam vários pacotes dirigidos ao mesmo porto de saída pode não ser possível realizar o seu encaminhamento imediato. Quando isso acontece, o *switch* necessita de colocar os pacotes que vão

chegando num *buffer* correspondente à porta de saída, para um posterior despacho. A arquitetura de um *switch* comum está ilustrada na figura 3.3, onde é possível também observar-se diferentes níveis de prioridade para as filas ou *buffers* das portas de saída. Estas filas correspondem a diferentes níveis de prioridade (8, máx), sendo dentro de cada fila o tráfego despachado de acordo com *First In First Out* (FIFO)/*First Come First Served* (FCFS) podendo originar na perda da utilidade da informação dos pacotes. Assim sendo, é possível verificar que mesmo com as vantagens que este tipo de configuração de rede proporciona, estas não são suficientes para prestar garantias de serviço e implementações mais eficientes terão que ser utilizadas para a assegurar a qualidade de serviço típico de um Sistema de Tempo-Real.

Por este motivo foram desenvolvidos diversos protocolos para implementar comportamentos tempo-real nas redes *Switched Ethernet*, podendo estes ser divididos em dois grandes grupos. Um deles está assente no uso de *switchs* de *Ethernet* comerciais ou COTS (*comercial of the shelf*) e o outro no uso de *switchs* modificados. No âmbito desta dissertação será necessário explicar um dos protocolos baseados no uso de *switchs* COTS, o *Flexible Time Triggered – Switched Ethernet* (FTT-SE) [27].

3.2 Flexible Time Triggered - Switched Ethernet

O FTT-SE é um protocolo para comunicações em rede *Switched Ethernet* para *switchs* COTS, seguindo o paradigma *Flexible Time Triggered*. Este protocolo propõe uma arquitetura *master multi-slave*, onde um só nó FTT *master* e diversos nós *slaves* encontram-se ligados a um *switch* COTS em topologia de estrela. Uma representação desta arquitetura pode ser vista na figura 3.4. O FTT *master* é responsável pela gestão e coordenação das comunicações existentes na rede, possuindo mecanismos de admissão de trafego, gestão de qualidade de serviço,

escalonador de trafego, base de dados para a reserva de *streams* (SRDB) e um *dispatcher*. Este encontra-se ligado a uma das portas do *switch* COTS pela qual realiza as suas operações através do envio de mensagens de controlo para os *slaves*. Desta forma, o FTT master controla o escalonamento das comunicações na rede, com o intuito de fornecer garantias de serviço na troca de pacotes entre os diversos nós *slave*. Este mecanismo de escalonamento de comunicações requer que todos os nós *slave* presentes na rede sejam compatíveis com o protocolo. Isto impossibilita o uso de nós *slave* do tipo *standard*, pois não possuem suporte para o protocolo. No entanto, este tipo de arquitetura permite que problemas já mencionados, como o consumo elevado de memória devido ao fluxo elevado de pacotes por uma porta do *switch* não ocorram, visto que todas as comunicações são instanciadas pelo FTT *master*.

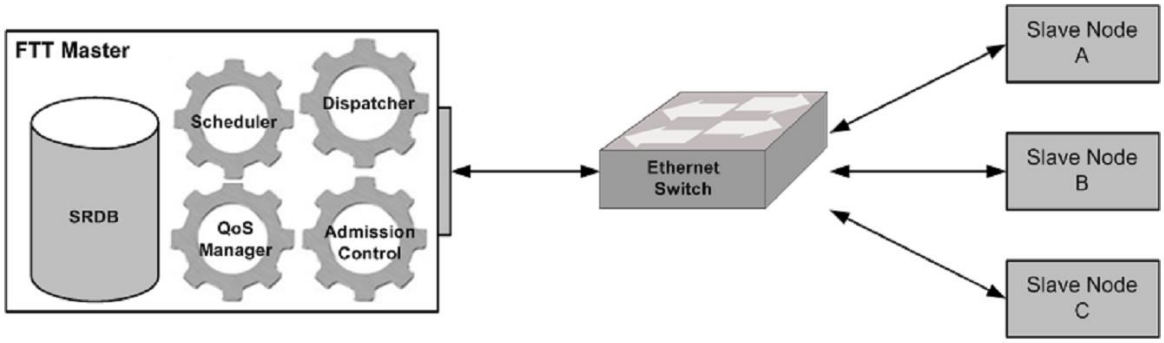

Figura 3.4: Arquitetura genérica de uma rede FTT-SE [25]

A comunicação por parte de todos os nós é estruturada, encontrando-se definido um ciclo elementar (EC) de duração fixa que é repetido ao longo do tempo. Neste EC, representado na figura 3.5, é possível verificar a existência de três janelas temporais para diferentes tipos de comunicação. O ciclo inicia com o envio de uma mensagem de controlo, denominada de *Trigger Message* (TM), por parte do FTT *master*, que será entregue a todos os outros nós da rede. Esta mensagem serve para sincronizar todos os nós *slave* e instanciar todos os momentos que cada um destes deverá comunicar. Após a receção e descodificação da TM por parte dos *slaves*, irá iniciar-se a janela temporal *Synchronous Window*,

dedicada à troca de mensagens periódicas, síncronas, e mais tarde iniciar-se-á a janela temporal *Asynchronous Window*, dedicada à troca de mensagens aperiódicas/assíncronas ou de mensagens que não necessitem de garantias de *QoS*, finalizando assim todas as etapas do ciclo. É necessário também referir que para além da duração do ciclo elementar (LEC) ser fixa, isto também é verdade para a duração da mensagem de controlo (LTM) e do *turn-around time* (TAT), fixado por configuração e que corresponde ao tempo de descodificação e processamento da resposta por parte dos nós *slave*.

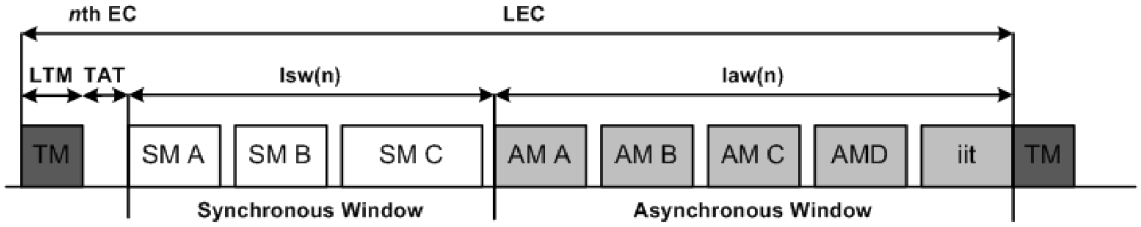

Figura 3.5: Arquitetura genérica de um ciclo elementar [25]

Durante a janela síncrona, irão ocorrer as trocas de mensagens periódicas entres os *slaves*. No FTT-SE cada envio de uma mensagem foi previamente instanciado pelo FTT master através da informação colocada na TM e enviada a todos os nós da rede. Desta forma, cada nó *slave* que se encontra agora sincronizado graças à mensagem TM, sabe a janela temporal onde irá comunicar.

Durante a janela assíncrona, irão ocorrer as trocas de mensagens aperiódicas entre os *slaves*. No entanto, para realizarem a comunicação deste tipo de tráfego, o nó em questão terá de transmitir ao FTT *master* estes pedidos durante a janela de controlo, para que o envio desta mensagem seja escalonado quando possível [28]. Na figura 3.6, é possível ver-se como funciona o mecanismo do pedido para a comunicação de tráfego assíncrono, e de seguida na figura 3.7 um exemplo de todos os tipos de comunicações na rede. É necessário referir que as janelas síncrona e assíncrona não possuem duração fixa. No entanto, a janela de tráfego síncrono possui uma duração máxima fixa, sendo que a variação dos valores temporais desta janela está inteiramente dependente da quantidade de

tempo necessária para a comunicação do tráfego periódico escalonado para esse EC. Esta duração máxima serve para garantir uma duração mínima para a janela de tráfego assíncrono.

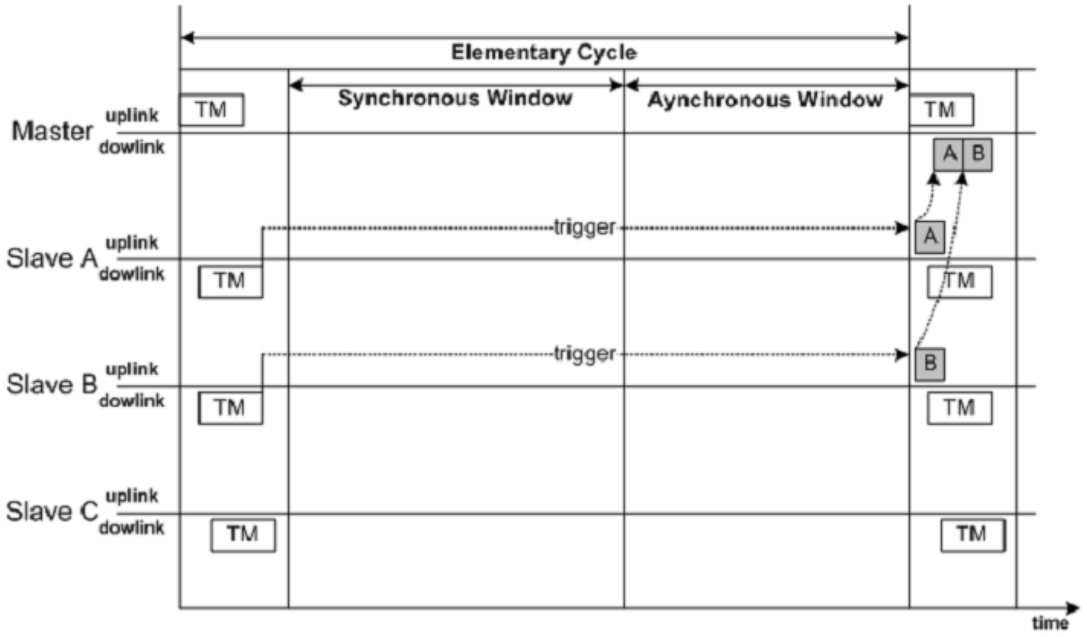

Figura 3.6: Sinalização de tráfego assíncrono no FTT-SE [25]

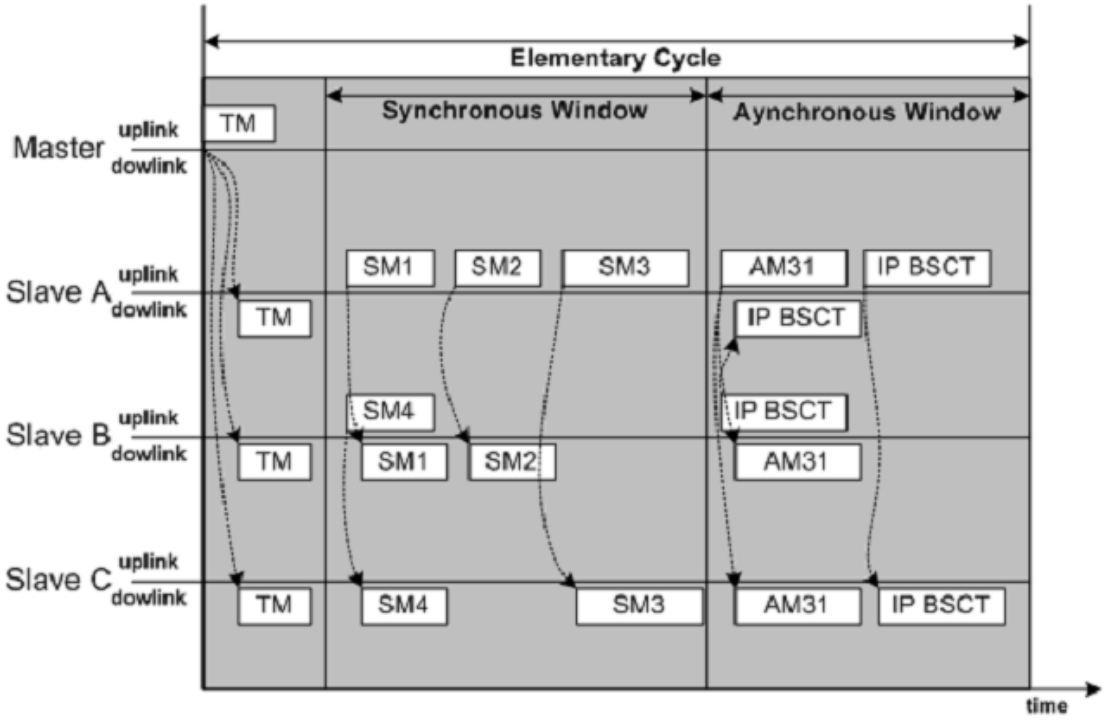

Figura 3.7: Sinalização de transmissões no FTT-SE [25]

3.3 HaRTES

No protocolo FTT-SE, todos os nós da rede têm que ser compatíveis com o protocolo. Isto é imperativo visto que é necessário uma boa sincronização das janelas temporais, pois a comunicação fora dos *slots* temporais atribuídos a cada nó pode comprometer o correto funcionamento. Isto impossibilita que nós que não possuem suporte para o protocolo FTT-SE sejam adicionados à rede. Sendo impossível de instalar drivers para compatibilidade do FTT a alguns nós, por exemplo devido a incompatibilidade com os sistemas operativos, foi necessário achar uma solução que ultrapasse esta limitação.

Segundo Santos [25], é possível ultrapassar esta limitação através do uso de um *switch* modificado e a introdução do FTT master dentro deste, representado na figura 3.8. Esta solução, desenvolvida na Universidade de Aveiro, dá origem ao *switch* HaRTES (*Hard Real-Time Ethernet Switch*), que possui a capacidade de controlar o tráfego à saída das suas portas, com ênfase deste controlo a nível temporal, realizando assim o *traffic shaping* para forçar a conformidade do tráfego recebido de acordo com o protocolo.

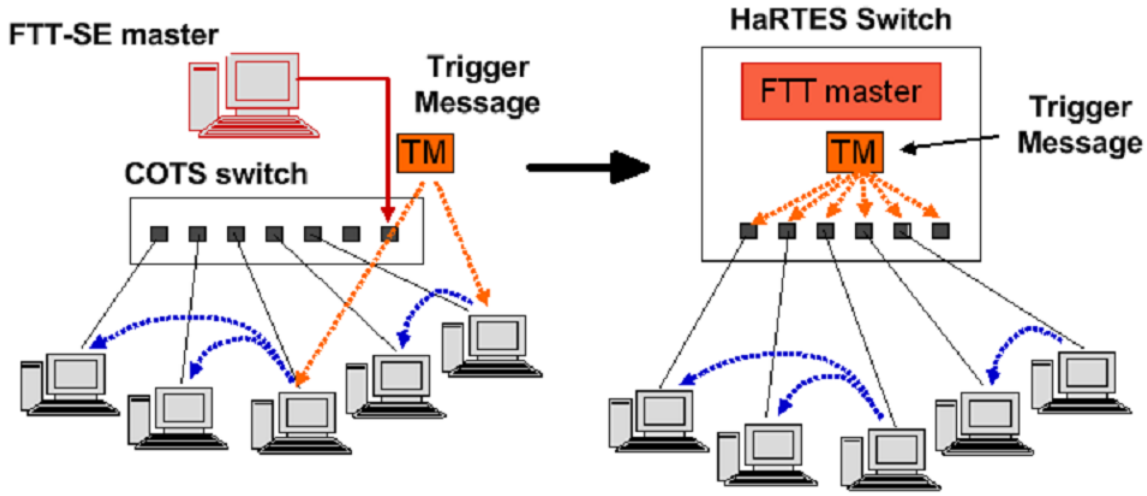

Figura 3.8: Criação do HaRTES, um *switch* desenvolvido para o FTT-SE [25]

Segundo Santos [25], esta solução, para além de permitir manter os aspetos mais essenciais do FTT-SE, também introduz diversas melhorias ao sistema de comunicação. Estas melhorias são:

- Aceitação de nós compatíveis e não compatíveis com o FTT, garantido a qualidade de serviço aos nós compatíveis ao mesmo tempo que se torna transparente os mecanismos de comunicação aos nós não compatíveis;

- Redução dos tempos de latência e *jitter* na transmissão das TM e melhoria na sincronização de todo o sistema, visto que o envio das mensagens TM têm agora origem no próprio *switch*;

- Aumento da integridade do sistema, visto que comunicações de tempo-real não desejadas podem ser diretamente bloqueadas nas portas do switch não envolvendo qualquer tipo de comunicação a um nó FTT master;

- Simplificação do mecanismo de comunicação de tráfego assíncrono por parte dos nós compatíveis com o FTT, visto que não será necessário qualquer pedido de envio desse tráfego, graças ao uso de filas de memória dentro do *switch* para tráfego assíncrono [25].

A simplificação da comunicação do trafego e aceitação de nós não compatíveis com o FTT, resultam da capacidade de controlo da saída de tráfego pelas suas portas. O *switch* HaRTES consegue este controlo guardando em filas de memória de tráfego todas as mensagens deste tipo que teriam como saída a porta correspondente a uma destas filas, sendo estas de nós compatíveis com o FTT ou não, transmitindo assim as mensagens guardadas em memória, apenas nas janelas temporais assíncronas. Embora este mecanismo seja transparente para os nós *standard*, para os nós compatíveis com o FTT será necessário o pedido de comunicação ao *switch*, com a passagem dos parâmetros correspondentes à duração da transmissão da *stream* e o seu mit. Este pedido, permite alocar

memória no *switch* HaRTES, garantido assim recursos suficientes para admitir todo o fluxo de mensagens dando garantias de serviço ao sistema.

4. SRP – Stream Reservation Protocol

Inicialmente foram introduzidos, no segundo capítulo, os conceitos que definem os Sistemas de Tempo-Real, estabelecendo o conceito de tarefa, de características e de classes de técnicas de escalonamento e em detalhe alguns algoritmos específicos que podem garantir a qualidade de serviço do sistema. De seguida, no terceiro capítulo, foi exposta a necessidade dos sistemas com uma arquitetura distribuída possuírem garantias de qualidade de serviço ao nível das suas comunicações. A este nível foi também explicado o esforço que tem vindo a ser desenvolvido para a criação de técnicas que possibilitem o uso da tecnologia Ethernet com garantias de tempo-real. Finalizando o terceiro capítulo, foi descrito o *switch* modificado HaRTES que é capaz de fornecer a sistemas distribuídos com comunicações através de Ethernet as garantias desejadas de tempo-real.

Neste último item, para além de ter sido desenvolvido o *hardware* que é denominado como *switch* HaRTES, também foi desenvolvido um protocolo para o mecanismo de reserva de recursos. Apesar do HaRTES ser capaz de aceitar tráfego de nós *standard* e fazer o despacho deste através das filas de memória de tráfego NRT, o mecanismo de reserva e sinalização não é normalizado, restringindo assim o uso deste *switch* a aplicações específicas FTT.

Foi então desenvolvido um esforço para encontrar um protocolo comercial para a sinalização de reserva de recursos, que fosse possível de ser integrado na plataforma HaRTES. Nóbrega [29], analisou os protocolos *Resource Reservation Protocol* (RSVP) e o *Stream Reservation Protocol* (SRP) e concluiu que o RSVP não

fornece as características necessárias para esta integração e sugere que o *Multiple Stream Reservation Protocol* (MSRP), um protocolo para a sinalização da reserva de recursos que faz parte do SRP, seja o protocolo escolhido para futuros trabalhos no *switch* HaRTES. Todos os protocolos, mecanismos e estruturas usadas no SRP encontram-se especificadas no *standard* IEEE STD 802.1Q™-2011 [30]. Esta sugestão foi realizada tendo em conta cinco componentes essenciais que, segundo *Zhang, Deering, Estrin, Shenker* e *Zappala* [31], uma arquitetura de rede terá de apresentar para ser capaz de fornecer aos seus nós garantias de qualidade de serviço. Nóbrega [29] expõe estas componentes essenciais, imprescindíveis para a sua sugestão, como sendo: a especificação do fluxo, encaminhamento, reserva de recursos, controlo de admissão e escalonamento de pacotes.

Característica/ Protocolo	*RSVP*	*MSRP*
Fonte do protocolo	*IETF*	*IEEE*
Serviços	*Controlled-Load e Guaranteed service*	8 diferentes prioridades
Estações intervenientes	Terminais e *routers*	*Bridges* , *switches* e terminais
Especificação de tempo de acesso	Não	Permite
Definição de *Deadlines*	Não	Permite
Latência acumulada	Exige	Exige
Origem das reservas	*Listeners*	*Listeners*

Figura 4.1: Distinção entre as características dos protocolos (adaptada de [29])

Na figura 4.1, Nóbrega [29] exibe as principais características que distinguem os dois protocolos. É possível entender-se que o RSVP foi desenvolvido para *routers* e não para bridges ou *switchs*. Além do mais, dentro dos Sistemas de Tempo-Real é imprescindível a especificação de restrições temporais. O RSVP encontra-se muito limitado a este nível, impossibilitando o uso desta definição e por consequência também a sua integração no *switch* HaRTES.

O SRP contém mais que um protocolo de sinalização, para estabelecer reservas para *streams* ao longo de uma rede implementada com switchs/*bridges*. Dentro do SRP existem então três protocolos, o *Multiple Mac Registration Protocol* (MMRP), o *Multiple VLAN Registration Protocol* (MVRP) e o MSRP. O MMRP é

usado para controlar a propagação de pedidos de reserva por parte dos *Talkers* através do registo de endereços MAC. O MVRP é utilizado por parte dos nós, ou segundo a norma as estações terminais, e as *bridges* para a declaração de *VLANs* dentro da rede. O MSRP como já foi referido antes, é um protocolo para a sinalização de reservas de recursos na rede. Este protocolo disponibiliza às estações terminais, nomeadamente os *Talkers*, a capacidade de reservar ao longo da rede até ao *Listener* desejado, os recursos necessários para garantir a qualidade de serviço desejada para as *streams* de dados que estes irão produzir.

Sendo o âmbito desta dissertação, a continuação do trabalho que tem vindo a ser realizado para introduzir no *switch* HaRTES suporte para o protocolo MSRP, será necessário explicar todo os mecanismos que este protocolo define.

4.1 MSRP – Multiple Stream Registration Protocol

O MSRP permite a cada estação reservar recursos destinados a um ou mais *Listeners* ao longo da rede. Esta reserva é realizada através de um pedido efetuado por parte de um *Talker*, que contém as características da *stream* a ser produzida e informação adicional sobre outras características da comunicação e do pedido de reserva. Este pedido é propagado ao longo da rede, com destino ao *Listener* desejado. No evento da chegada de um pedido de reserva, o *Listener* terá de responder se está ou não disponível para receber a *stream*, e irá proceder ao envio da resposta com destino ao *Talker*. Durante esta troca de mensagens, o pacote concebido para este pedido de reserva de recursos, será analisado por cada *bridge* por onde passe. Cabe assim à *bridge* controlar o fluxo destes pacotes, tanto ao nível de reencaminhamento como ao nível de controlo de admissão de *streams*. Assim sendo, caso um pedido de reserva passe por uma *bridge* que, e esta após analisar o pacote e os recursos disponíveis, não disponha de recursos suficientes para a garantir a qualidade de serviço desejada, a *bridge* deverá informar o *Talker* em questão que o pedido de reserva falhou. Para realizar todas estas atividades, o

protocolo define duas aplicações *Multiple Registration Protocol* (MRP) [30]: o *MRP Attribute Declaration* (MAD) [30] e o *MRP Attribute Propagation* (MAP) [30].

4.1.1 MRP - Multiple Registration Protocol

De acordo com a norma IEEE STD 802.1Q™-2011 [30], o MRP permite a uma aplicação MRP realizar ou libertar declarações de atributos resultando no registo ou na eliminação de um registo desses atributos noutros participantes MRP. A declaração de um atributo por parte de um participante MRP é registada pela máquina de estados de declarações (*Applicant state machine*) dessa aplicação designada para esse atributo, sendo que independentemente da ação a realizar, qualquer alteração de uma variável produz a comunicação de *Multiple Registration Protocol Data Units* (MRPDUs) para anunciar a declaração. As estações terminais de destino das MRPDUs e as portas das *bridges* intermediárias para a comunicação destas, gravam o registo dos atributos nas suas máquinas de estados de registos (*Registrar state machine*), à medida que estas mensagens chegam. A figura 4.2, pretende ilustrar o resultado de uma única estação terminal a realizar uma declaração, e a propagação desta pelas portas das *bridges* intermediárias. É possível verificar a declaração inicial *D* apontada pela seta a vermelho e o registo *R* desta na porta da *bridge* mais próxima que irá resultar na propagação de *D* para as suas restantes portas. O comportamento repete-se ao longo da rede (*flooding* da rede).

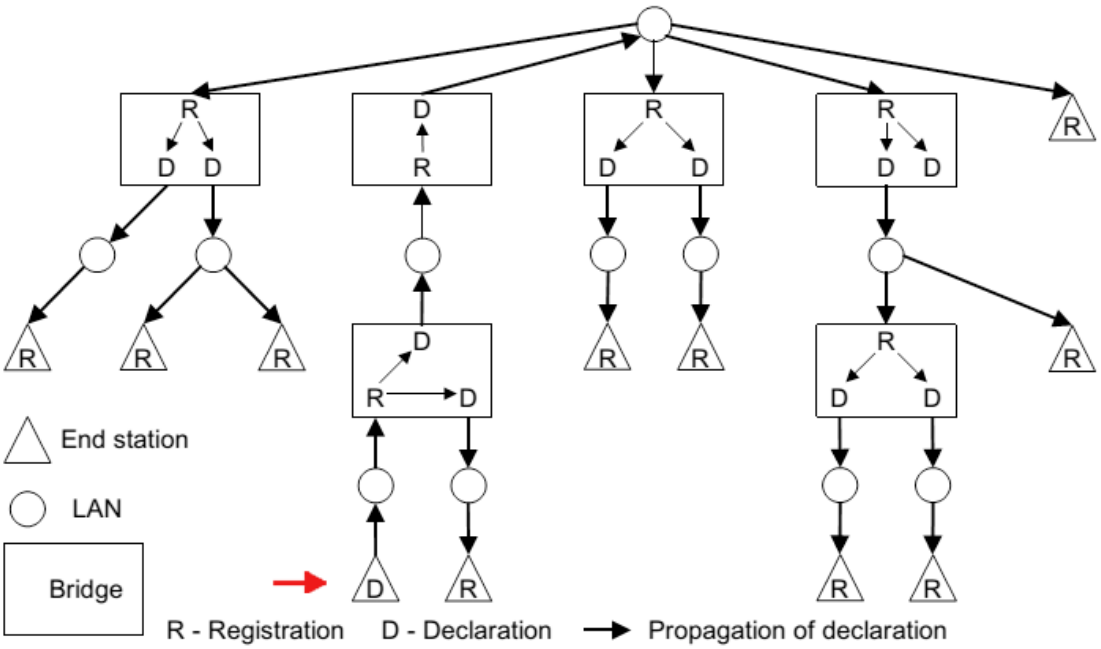

Figura 4.2: Exemplo da propagação de atributos de uma estação terminal (adaptado de [30])

4.1.1.1 Arquitetura MRP

Um participante MRP é composto pela sua componente própria de aplicação individual e pelo seu MAD. A componente de aplicação irá tratar de gerar toda a semântica associada aos valores de cada atributo e o seu consequente registo. Para isto a aplicação lança um pedido ao seu MAD, podendo utilizar uma das duas primitivas existentes para realizar ou desfazer declaração, sendo estas: *MAD_Join.request(attribute_type, attribute_value, new)* e *MAD_Leave.request(attribute_type, attribute_value)* [30]. O campo *attribute_type* especifica o tipo de atributo associado a esta declaração, o *attribute_value* especifica a instância desse tipo, e por fim a variável booleana *new* especifica que se trata de uma declaração nova. De seguida a componente MAD responsável pela correta propagação de mensagens descrita no protocolo MRP, gera as MRPDUs a transmitir e processa as mensagens recebidas de outros participantes, comunicando à componente de aplicação a alteração de um registo de atributo

através das primitivas: *MAD_Join.indication(attribute_type, attribute_value, new)* e *MAD_Leave. indication (attribute_type, attribute_value)* [30].

As *bridges*, para além das componentes presentes num participante MRP, também possuem uma componente MAP que irá controlar a propagação de informação entre portos participantes, usando as mesmas primitivas referidas no parágrafo anterior. Na figura 4.3, encontram-se representadas as componentes de participantes MRP entre uma *bridge* de duas portas e uma única estação terminal.

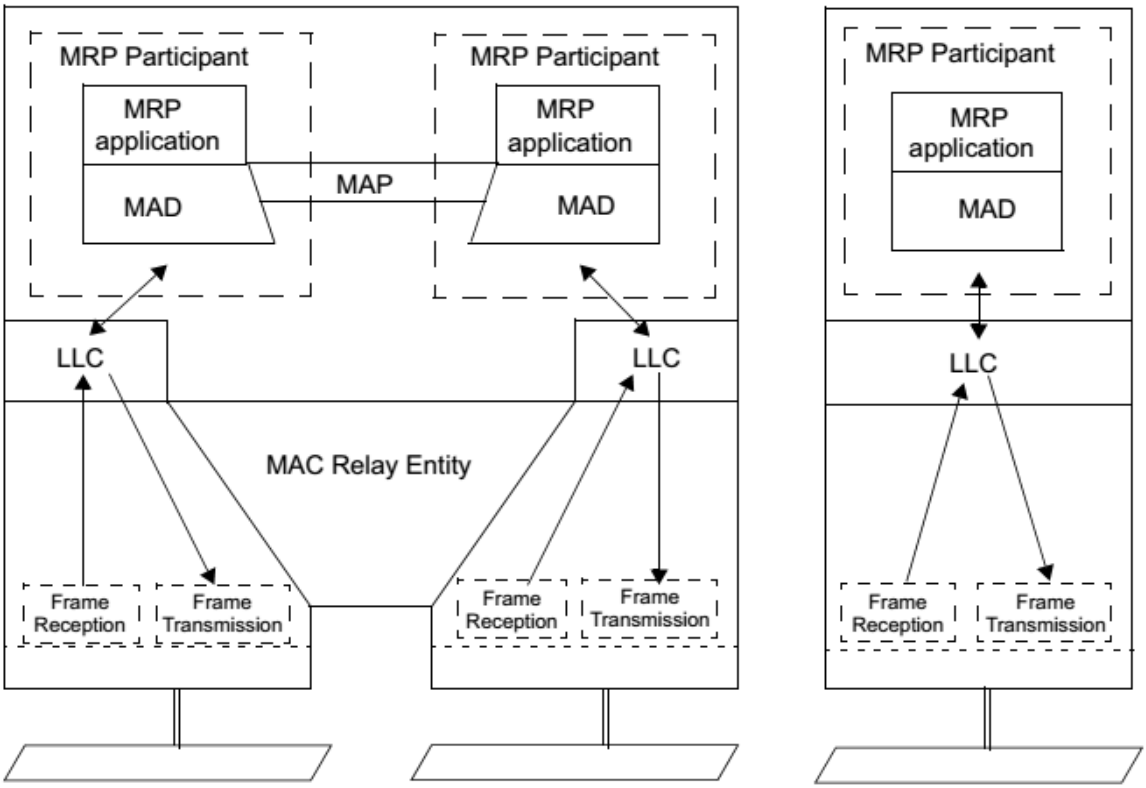

Figura 4.3: Arquitetura MRP [30]

4.1.1.2 MRPDUs – MRP Protocol Data Units

Dentro de uma rede com a arquitetura MRP, todas as mensagens trocadas entre os participantes MRP apresentam a codificação e estrutura das MRPDUs. Dentro de cada MRPDU é possível, através da informação nela contida, identificar em que aplicação MRP esta foi gerada e qual é a aplicação de destino. Estes campos são imprescindíveis para mecanismos de *fowarding* que poderão ser

aplicados. Cada MRPDU pode conter, uma ou mais mensagens MRP, cada uma delas identificando um ou mais eventos do domínio MRP e a classe dos atributos correspondentes a tais eventos. Na figura 4.4, é possível verificar-se a estrutura de uma MRPDU, onde é de salientar que embora os campos sejam preenchidos de diversas formas consoante a aplicação, estas apresentam sempre a estrutura genérica ilustrada nesta figura. É possível também verificar-se que todas as MRPDUs que seguem o protocolo IEEE STD 802.1Q™-2011 [30], consistem num número inteiro de octetos sendo que o primeiro octeto possui a versão do protocolo seguido de uma ou mais mensagens. Dentro da estrutura da mensagem é possível verificar-se a presença dos campos *AttributeType*, *AttributeLength* e *AttributeList*. Dentro da lista de atributos existe uma estrutura composta por um ou mais *VectorAttribute*, sendo que o último elemento será o terminador *EndMark*. Um *VectorAttribute* é constituído por um *VectorHeader*, um *FirstValue*, e um *Vector*. Este *VectorHeader* é constituído pelo *LeaveAllEvent* e o número de eventos de atributos codificados no *Vector*.

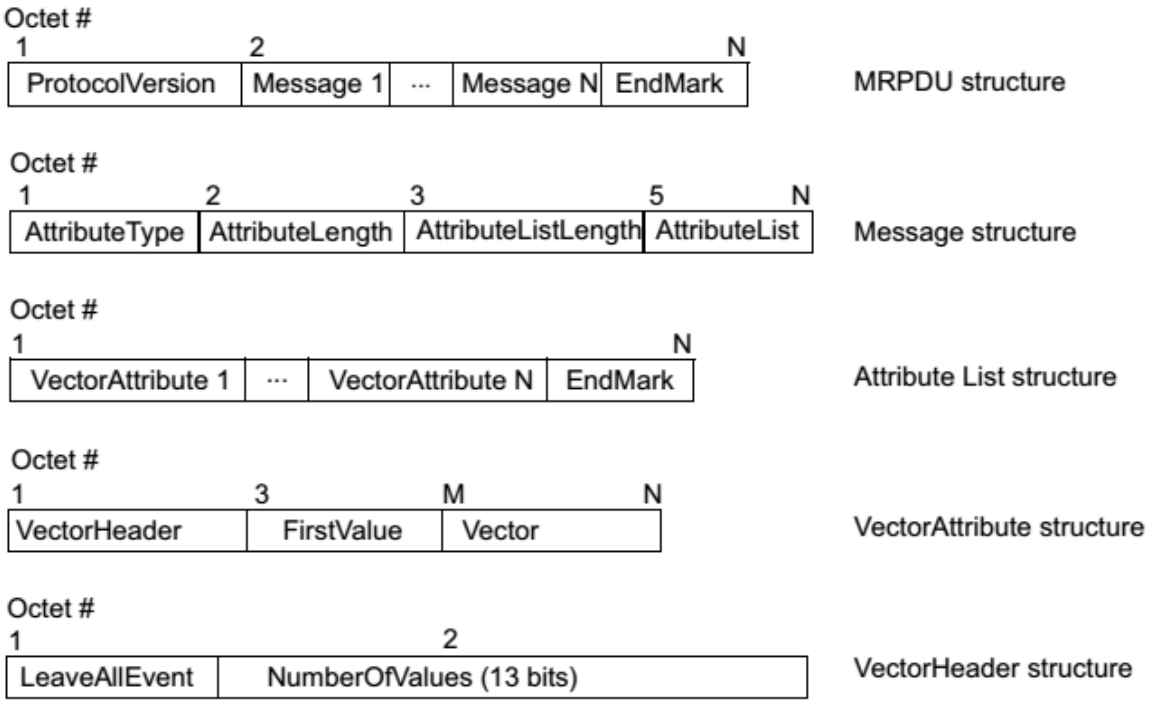

Figura 4.4: Formato das componentes principais de uma MRPDU [30]

4.1.1.3 MAD – MRP Attribute Declaration

O MAD, presente em todas instâncias de um participante MRP ao longo de uma rede com a arquitetura MRP, apresenta um nível elevado de importância, pois é esta componente que é capaz de gerar as mensagens MRPDUs ao mesmo tempo que possui as máquinas de estado que mantêm os estados dos atributos ao longo da rede. Devido a esta elevada importância, tanto no protocolo MRP como no MSRP, é necessário de uma forma mais aprofundada explicar certas características que esta componente possui.

Como mencionado anteriormente a execução do MRP é realizada pelo MAD, que para preservar os estados dos atributos ao longo de uma rede possui um certo conjunto de máquinas de estados, estas são:

- **Applicant State Machine por atributo,** diretamente associada à interação com a componente de aplicação do participante MRP, tendo desta forma uma função de controlo;

- **Registrar State Machine por atributo**, diretamente associada com a comunicação com os outros participantes MRP, preservando todas as declarações de atributos vindas da rede;

- **LeaveAll State Machine por cada participante MRP,** utiliza uma mensagem única, aplicada a todos os atributos, que resulta na destruição do registo obrigando a que todos os participantes lancem novas declarações;

- **PeriodicTransmission State Machine por cada participante MRP**, disponibiliza a opção de envios periódicos de declarações, assegurando que o registo se mantém ativo.

Apesar de todos os MADs possuírem as mesmas máquinas de estado, a utilização ou não destas por parte do participante MRP cria quatro implementações diferentes que poderão caracterizar o participante. Estas quatro implementações são [30]:

- **Full Participant,** quando este implementa a *Applicant* e a *Registrar state machine* para cada um dos atributos declarados, registados, ou monitorizados, assim como uma *LeaveAll* e uma *PeriodicTransmission state machine*;

- **Point-to-point subset of Full Participant**, que em relação ao *Full Participant* este apenas omite certos estados e ações das *Applicant state machines*;

- **Applicant-Only Participant,** quando este implementa apenas a *Applicant state machine* para cada um dos atributos declarados, registados, ou monitorizados, omitindo algumas ações e estados desta, juntamente com uma *PeriodicTransmission state machine*;

- **Point-to-point subset of Applicant-Only Participant**, que em relação ao *Applicant-Only Participant* omite ações e estados adicionais.

4.1.1.4 MAP – MRP Attribute Propagation

Como mencionado anteriormente, o MAP possibilita a propagação das declarações de atributos dentro de uma *bridge* entre as suas portas, e por consequência ao longo da rede e entre os diversos participantes MRP. A esta propagação poderá também estar associada uma admissão seletiva das mensagens a serem propagadas por certas portas da *bridge*, nomeadamente no MSRP, que será analisada nesta dissertação. No entanto esta admissão seletiva terá de ser desenvolvida na própria *bridge*, pois para o protocolo MRP, esta característica não está contemplada. Uma característica na propagação que é necessário salientar é que dentro da *bridge*, todas as portas possuem um *Fowarding State* que caso inativo impossibilita a propagação de mensagens através dessa porta.

4.1.2 Declarações de Atributos no MSRP

Como foi previamente analisado nos subcapítulos destinados ao MRP, este protocolo possui uma arquitetura com várias estações terminais e uma ou mais *bridges*. No MRP, cada estação terminal, ou cada porta da *bridge* possuía em si um módulo *MRP Participant*, sendo este composto por um MAD e uma aplicação MRP. Este MRP participante iria, como foi mencionado, gerar declarações de atributos para posterior propagação na rede.

Transpondo isto para o MSRP, temos agora uma arquitetura semelhante constituída por estações terminais que podem tomar o papel de *Talker* ou de *Listener*, e *bridges*. Nesta arquitetura as declarações de atributos têm origem nos *Talkers*, podendo estes realizar apenas dois tipos de declarações, o *Talker Advertise* e o *Talker Failed*. A declaração *Talker Advertise* emitida por um *Talker*, especifica a vontade de estabelecer uma reserva na rede para uma determinada *stream* a comunicar. Quando emitida, esta será propagada pela rede muito em semelhança ao método utilizado no protocolo MRP, atingindo as restantes estações terminais. Após a descodificação desta declaração por parte das restantes estações, se uma destas pretender receber a *stream* identificada no *Talker Advertise*, passa a tomar um papel de *Listener* e irá emitir um *Listener Ready*. Esta declaração, caso chegue ao *Talker* irá sinalizar-lo de que foram efetuadas reservas na rede que garantem a qualidade de serviço desejada, e que este pode iniciar a transmissão da sua *stream*. Ao longo da rede, as declarações *Talker Advertise* e *Listener Ready*, ao passar pelas *bridges*, serão descodificadas pelos respetivos MADs para posterior verificação por parte do MAP, da existência ou não de recursos disponíveis que suportem a qualidade de serviço desejada para esta *stream*. Caso estes recursos estejam disponíveis, os MAPs irão realizar a propagação de atributos pelas suas portas ao longo da rede. Na eventualidade destes recursos não se apresentarem, os MAPs possuem capacidades para gerar novas declarações nos módulos MAD das portas das suas *bridges*. Estas novas declarações são:

- **Talker Failed**, emitidas na direção dos *Listeners* na situação da chegada de um *Talker Advertise* a uma *bridge* sem recursos adequados;

- **Listener Asking Failed**, emitidas na direção dos *Talkers* na situação da chegada um *Listener Ready* a uma *bridge* sem recursos adequados;

- **Listener Ready Failed**, emitidas na direção dos *Talkers* na situação da chegada de dois ou mais *Listener Ready* a uma *bridge* que só é capaz de disponibilizar recursos para um destes *Listeners*.

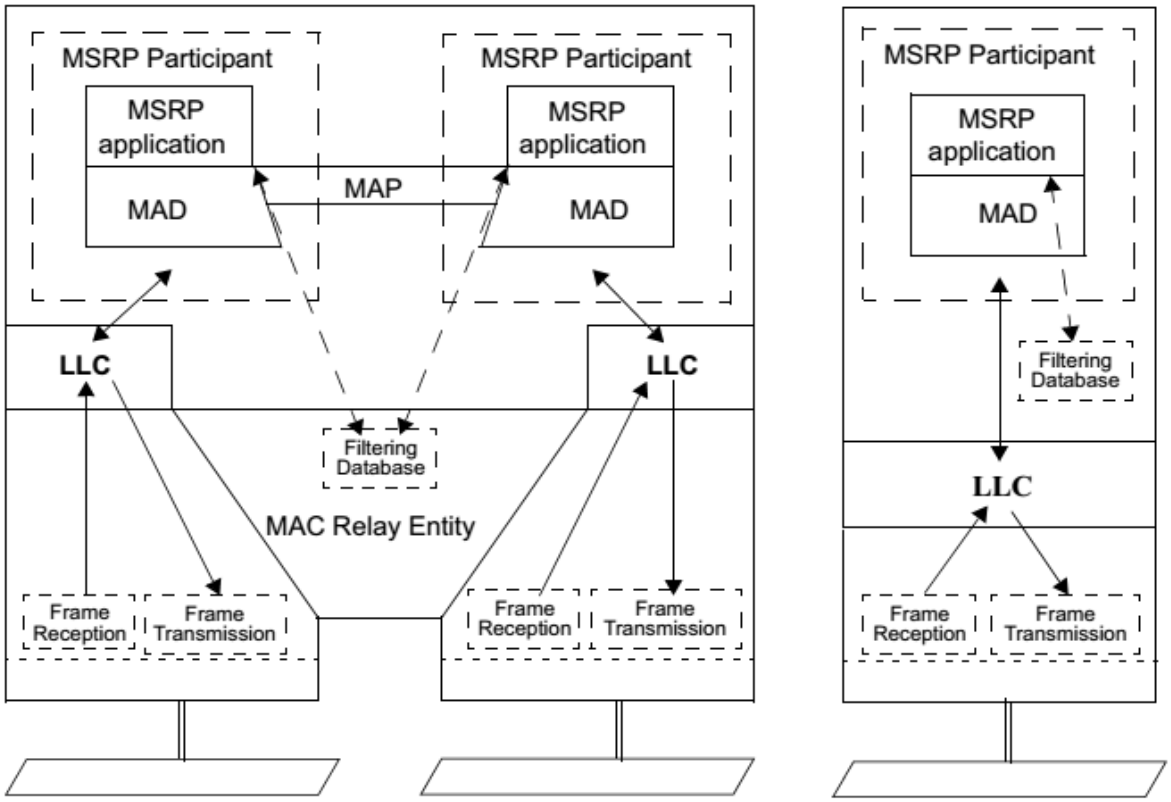

Figura 4.5: Arquitetura MSRP com uma *bridge* de duas portas e uma estação terminal [30]

Na figura 4.5, é possível verificar-se a arquitetura MSRP para o caso de uma *bridge* com duas portas e uma estação terminal. É de evidenciar que esta arquitetura é extremamente semelhante à do MRP. No entanto é possível concluir que com a adição de um controlo de admissão mais avançado, capacidade de reserva de recursos, encaminhamento e escalonamento de pacotes o MSRP

encontra-se capaz de realizar o controlo de admissão de tráfego desejada nos Sistemas de Tempo-Real.

4.1.3 MSRP application

A aplicação MSRP, em semelhança ao que acontece no MRP, está encarregue da semântica das declarações de atributos e dos registos imprescindíveis para o correto funcionamento do protocolo. Para isso mantem assim os dois tipos de variáveis, o primeiro tipo será as variáveis usadas internamente pelas *Application state machines* sendo que o segundo será as variáveis transportadas através das MSRPDUs, que possuem a estrutura já previamente analisada das MRPDUs. Um dos campos de interesse para o MSRP é o *FirstValue*, onde são trocados os atributos do MSRP.

As variáveis internas e parâmetros utilizados pelas máquinas de estado são:

- **Port Media Type**, variável que define como é feito o processamento das MSRPDUs segundo o acesso ao meio;

- **Direction**, variável que identifica se uma declaração de atributos corresponde a uma declaração por parte de um *Talker* ou de um *Listener*;

- **Declaration Type**, indica o tipo específico de uma declaração por parte de um *Talker* ou de um *Listener*;

- **SRP Parameters**, que são um conjunto de parâmetros que estabelecem valores relacionados com os recursos da rede (*portTcMaxLatency*, *talkerPruning*, *streamAge*, *msrpEnabledStatus*, *msrpMaxFanInPorts*, *msrpLatencyMaxFrameSize*, *SRPdomainBoundaryPort*, *SR_PVID*) [30].

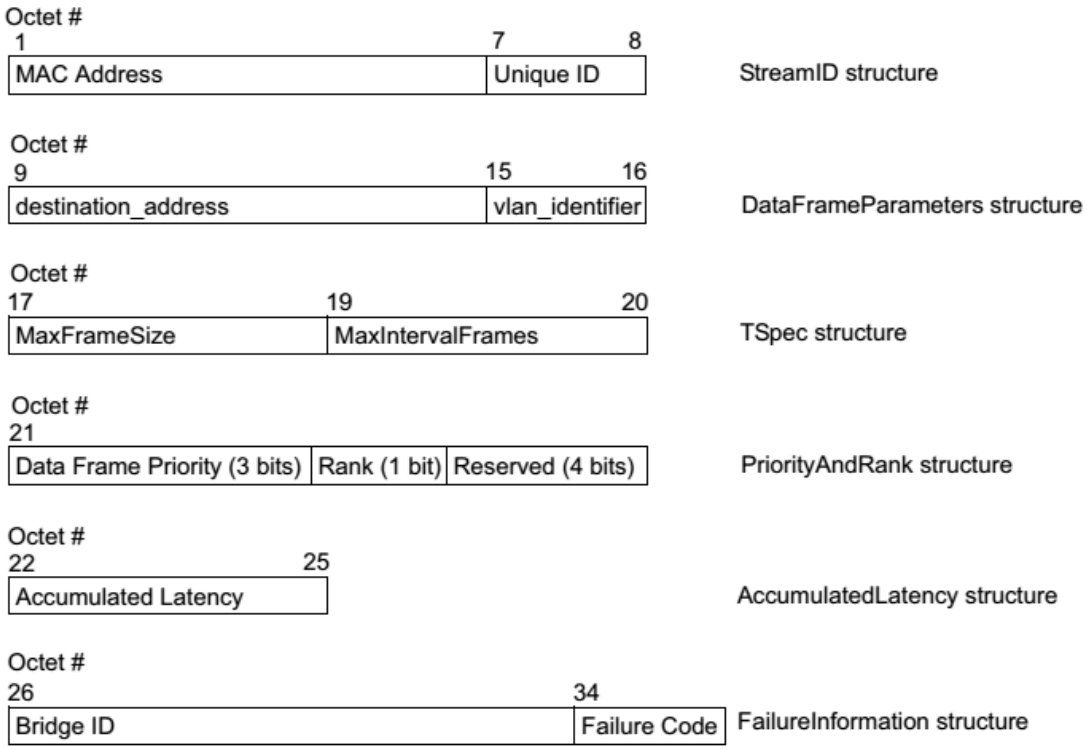

Figura 4.6: Formato das componentes principais do FirstValue [30]

As variáveis trocadas referentes aos atributos do MSRP, podem ser visualizadas na figura 4.6 que explicita o formato do campo *FirstValue* do MSRP. No âmbito desta dissertação, dentro campo *FirstValue* é possível destacar dois tipos de variáveis importantes para o trabalho realizado, as de carácter funcional para o sistema e as de carácter de tempo-real. Dentro das de carácter funcional temos toda a estrutura do *StreamID* e do *DataFrameParameters*. Dentro das de carácter de tempo-real temos toda a estrutura *TSpec* e *PriorityAndRank*. Os campos essenciais do *FirstValue* são:

- **Unique ID**, identifica a *stream*;

- **MaxFrameSize,** corresponde ao tamanho máximo do pacote que o *Talker* irá produzir, excluindo qualquer *overhead*;

- **MaxIntervalFrames,** corresponde ao máximo número de pacotes que o Talker poderá transmitir;

- **Data Frame Priority,** determina em que fila da porta de saída os pacotes da *stream* serão colocados;

- **Rank,** sinaliza a *stream* como urgente;

- **Accumulated Latency,** é utilizado para calcular o pior caso de latência que uma *stream* pode encontrar no seu percurso entre o *Talker* e um determinado *Listener*.

4.1.4 MAP no MSRP

Dotado de uma maior capacidade de análise em relação ao protocolo MRP, o MAP no MSRP é responsável pelo controlo e propagação de declarações de atributos ao longo da rede. Como mencionado anteriormente, nas *bridges* do MSRP face à chegada de declarações de atributos quer por parte de um *Talker* ou de um *Listener*, esta necessita de realizar um controlo destas mensagens. Para isto o MAP necessita de comparar as variáveis descodificadas pelo MAD do campo *FirstValue* com as capacidades que os recursos presentes nessa *bridge* apresentam no momento da chegada destas mensagens. Para esta comparação o MAP irá fazer uso da base dados descrita na arquitetura do MSRP, e caso conclua que a *bridge* apresenta condições para assegurar a qualidade de serviço decretada pelos parâmetros *TSpec* irá propagar as declarações recebidas em direção aos seus destinos. No caso de concluir que por algum motivo não será possível prestar as garantias desejadas, o MAP necessita de comunicar com os módulos MSRP *participant* das portas da *bridge* correspondentes ao *Talker* e aos *Listeners* que criem novas declarações que reportem que a reserva de recursos não poderá ocorrer.

Estas novas declarações e todo o comportamento de controlo foi explicitado no capítulo 4.1.2. No momento em que o MAP recebe um *Listener Ready*, e a análise dos recursos concluir que a reserva é possível, este deverá para além de

propagar as declarações desejadas, reservar também os recursos físicos que serão utilizados para comunicação da *stream* a ser produzida pelo *Talker*.

5. Implementação

O trabalho desenvolvido nesta dissertação pretende fornecer ao *switch* HaRTES todas as capacidades necessárias para conseguir suportar as funcionalidades de reservas de recursos decretadas no protocolo MSRP. Sendo assim neste capítulo, pretende-se introduzir e explicar o conjunto de implementações que foram concebidas para integrar no *switch* HaRTES, com o propósito de completar o mecanismo de reservas de recursos segundo o MSRP. Inicialmente irá ser analisado o estado do *switch* HaRTES e o trabalho realizado previamente.

5.1 Trabalho Realizado Previamente

Para apresentar o trabalho realizado no âmbito desta dissertação, será necessário primeiro descrever as necessidades que o *swicth* HaRTES apresenta para ser compatível com os pedidos de reserva realizados segundo o MSRP, e descrever o estado das aplicações realizadas previamente a esta dissertação.

Como foi explicitado no capítulo anterior, o protocolo MSRP possui dois grandes módulos principais que asseguram a implementação deste protocolo numa *bridge*. Estes módulos principais são:

- **Módulo MSRP participant**: Este módulo, presente tanto nas estações terminais como nas portas das *bridges* intermediárias de uma rede MSRP, é constituído por dois sub-módulos o MAD e o *MSRP Application*. O sub-módulo

MSRP Application é responsável pela semântica dos atributos presentes nas declarações do MSRP, e o sub-módulo MAD é responsável pela gestão das máquinas de estados, codificação e descodificação da unidade básica de comunicação no protocolo, as MSRPDUs.

- **Módulo MAP**: Este módulo que está presente apenas nas *bridges* de uma rede MSRP é responsável pela gestão e propagação das declarações pelas respetivas portas da *bridge*. Para assegurar que a propagação das declarações que pretendem realizar reservas na rede, só ocorre quando existem condições para garantir a QoS solicitada este módulo tem de possuir a capacidade de verificação de diversas características, tanto dos seus recursos, como os da própria rede.

Pretendendo que o *switch* HaRTES assuma o papel de uma *bridge* característica do protocolo MSRP, é necessário então a implementação de um MAD e de um MAP, fornecendo assim ao HaRTES a capacidade de codificação e descodificação de mensagens MSRPDUs, e de admissão de reservas segundo o MSRP.

5.1.1 MAD

Inicialmente, para fornecer ao HaRTES a capacidade de gerir as mensagens relativas ao protocolo MSRP, foi desenvolvido uma aplicação capaz de produzir mensagens segundo o formato das MSRPDUs [29]. Com esta aplicação seria então possível instanciar um *Talker Advertise*, fornecendo toda a semântica relacionada com a declaração, que iria codificar uma MSRPDU correspondente a este pedido. Foi desenvolvido com este intuito uma aplicação então capaz de traduzir os pedidos das declarações *Talker Advertise*, *Talker Failed*, *Listener Ready*, *Listener Asking Ready* e *Listener Asking Failed*, em mensagens protocolares MSRPDU.

Seguindo o formato das MRPDUs, visível na figura 5.1, cujo formato das MSRPDUs é igual, a aplicação é capaz gerar uma mensagem protocolar pronta

para ser comunicada no campo de dados úteis da *frame* de um pacote de *Ethernet*. Apesar de o protocolo apresentar cinco tipos de declarações possíveis, apenas três tipos de atributos (*Talker Advertise, Talker Failed* e *Listener*) são utilizados para a codificação dos pedidos.

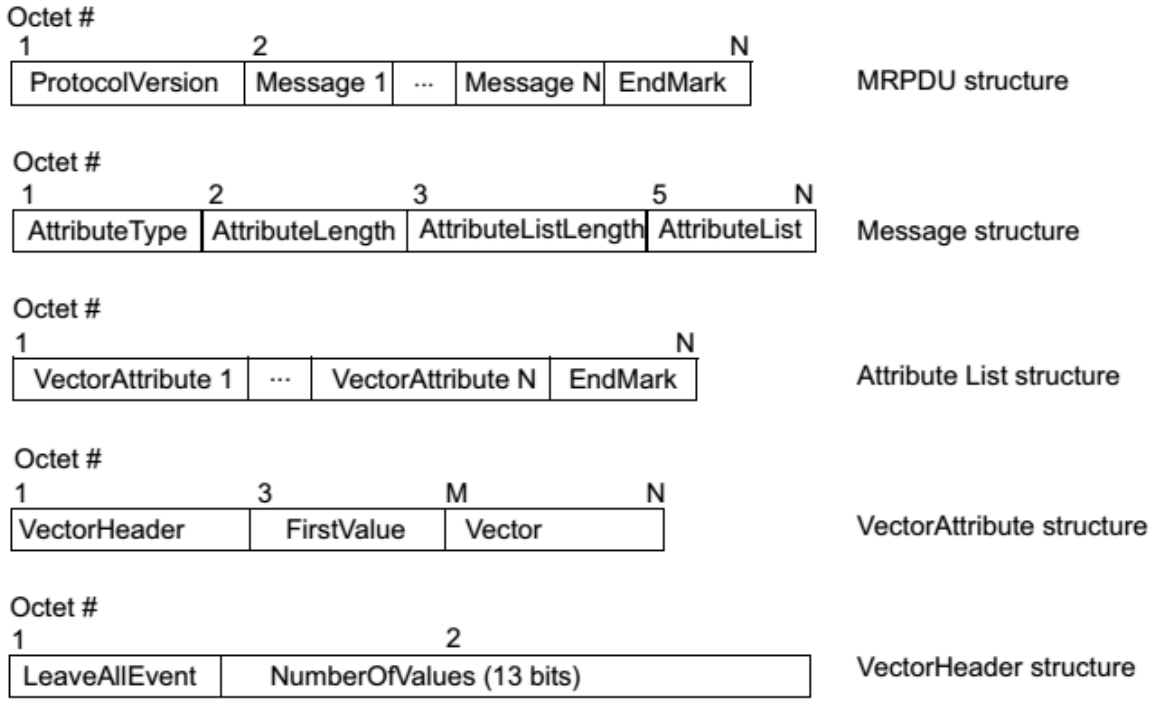

Figura 5.1: Formato das componentes principais de uma MRPDU [30]

Será então necessário explicitar o conjunto de campos que são preenchidos, começando pelos que são comuns a todos os tipos de mensagens protocolares MSRP, pois servem de identificadores do atributo e protocolo codificado dentro do pacote de *Ethernet*:

- **MSRP application address** (6 Bytes): preenchidos com o endereço MAC identificador do protocolo MSRP (01-80-C2-00-00-22);

- **EtherType** (1 Byte): preenchido com o identificador do protocolo MSRP que é encapsulado no campo de dados úteis da *frame Ethernet* (22-EA);

- **ProtocolVersion** (1 Byte): preenchido com a versão do protocolo correspondente;

- **AttributeType** (1 Byte): preenchido com um dos identificadores dos três tipos de atributos possíveis para uma mensagem MSRPDU (*Talker Advertise* – 1, *Talker Failed* – 2, *Listener* – 3);

- **AttributeLength** (1 Byte): preenchido com o número de bytes correspondente ao tamanho ocupado pelo atributo (*Talker Advertise* – 25 ou 0x19; *Talker Failed* – 34 ou 0x22; *Listener* – 8)

Para codificar os restantes tipos de declarações, nomeadamente as que pertencem às declarações relativas ao atributo *Listener*, é usada a estrutura *FourPackedEvents* no campo *Vector*, que irá definir qual a declaração realizada por parte do *Listener* (*Listener Asking Failed* – 1, *Listener Ready* - 2, *Listener Ready Failed* – 3). No caso de a declaração ser do domínio de um Talker, o campo Vector toma a estrutura de *ThreePackedEvents*, declarando o evento que se pretende realizar nas máquinas de estados do MSRP (*New* - 0, *JoinIn* - 1, *In* - 2, *JoinMT* - 3, *Mt* – 4, *Lv* - 5). O número destes eventos codificados numa só mensagem pode variar, sendo para isso usado o campo *VectorHeader*, composto pelo bit corresponde à existência de um *LeaveAllEvent* e a variável *NumberOfValues* que explicita o número de eventos codificados.

Será também necessário voltar a explicitar que toda a informação relativa às características da reserva da *stream* são colocados no campo *FirstValue*, já detalhado anteriormente. Uma consideração a ter em relação a estes campos que formam o *FirstValue*, é que o campo *StreamID*, que é preenchido para os três tipos de atributos. Os campos *DataFrameParameters*, *TSpec*, *Priority and Rank* e *Acumulated Latency* são preenchidos para as mensagens *TalkerAdvertise* e *TalkerFailed*, e o campo *Failure Information* apenas para a mensagem *TalkerFailed*.

Com o desenvolvimento desta aplicação tornou-se possível criar um cenário de teste onde duas estações terminais, simbolizando um *Talker* e um *Listener*, embora sem a *bridge* característica do MSRP entre eles, a realizarem através da Ethernet comunicações protocolares. É possível verificar na figura 5.2, os resultados obtidos num dos testes realizados, onde é impresso na consola da aplicação residente no *Listener* os dados retirados da chegada de um *TalkerAdvertise*, demonstrando a codificação, envio, receção e descodificação de MSRPDUs por parte da aplicação [29].

```
************* Ready do receive packets *************
Received packet from: 1D924770CB,
AttributeType encoded: 0X1 ->   Talker Advertise
Decoding Talker Advertise Packet..........

Protocol Version: 0
Attribute Type: 1
Attribute Lenght 19
Attribute List Lenght: 1E
Leave All Event: 0
Number of values/events:  1
Talker MAC Address: 1D924770CB
Stream Unique ID: 5B95
Stream Destination Address: F12A623C2565
Vlan ID: 86BD
TSpec Bandwith: 23587
TSpec Frame Rate: 23458
Traffic class: 2
Rank: 1
Reserved: 0
Acummulated Latency: 3494569216
Three Packed Event Byte: 0
EndMark1: 0
EndMark2: 0
```

Figura 5.2: Parâmetros de um TalkerAdvertise [29]

5.1.2 MSRP Application

Possuindo a capacidade de codificar e descodificar as MSRPDUs, o próximo passo lógico seria implementar na aplicação as máquinas de estados características do MAD e o *MSRP Application*. No entanto o objetivo dos esforços realizados serve para dotar o *switch* HaRTES de suporte para a reserva de recursos

segundo o protocolo MSRP e não implementar, como por exemplo, todas as componentes necessárias para o funcionamento das estações terminais.

Desta forma foi desenvolvido um *MRPDaemon* com as máquinas de estado e mecanismos necessários para preservar as características das reservas das *streams* na rede, nomeadamente os sinais *keepAlive* que asseguram que as reservas se mantenham na rede, juntamente com a geração e descodificação das mensagens protocolares desenvolvida por Nóbrega [29]. Este *MRPDaemon* foi concebido para a integração dentro do HaRTES, sendo criado uma instância *MRPDaemon*, em semelhança ao protocolo MSRP, em cada uma das portas do *switch*.

5.1.3 MAP

Adicionalmente foi também criado uma modificação do *MRPDaemon*, resultando na aplicação *MRPBridge*. Esta modificação foi projetada com o intuito de manter uma separação clara das camadas do *switch* correspondente ao funcionamento do FTT-SE, da camada correspondente ao funcionamento do MSRP. Convém mencionar que esta aplicação *MRPBridge* apresenta, adicionalmente às funcionalidades do *MRPDaemon*, uma base de dados que caracteriza as *streams*, os seus pedidos de reserva e o estado em que estes pedidos se encontram. Desta forma a aplicação funcionará um pouco diferente do que se pretende do MAP, embora capaz de realizar a propagação das declarações para os restantes MRPDaemons. Na figura 5.3 é possível visualizar a arquitetura que foi desenvolvida com a integração das duas aplicações criadas, sendo o MAD na figura correspondente ao MRPDaemon e a *Bridge Control* correspondente a MRPBridge. Esta figura exemplifica a arquitetura para um switch HaRTES dotado de duas portas em que cada uma se encontra conectada ao seu MRPDaemon correspondente. Adicionalmente ambos MRPDaemons encontram-se conectados

ao MRPBridge. Convém também mencionar que foram utilizadas redes virtuais para efetuar a conexão destes módulos.

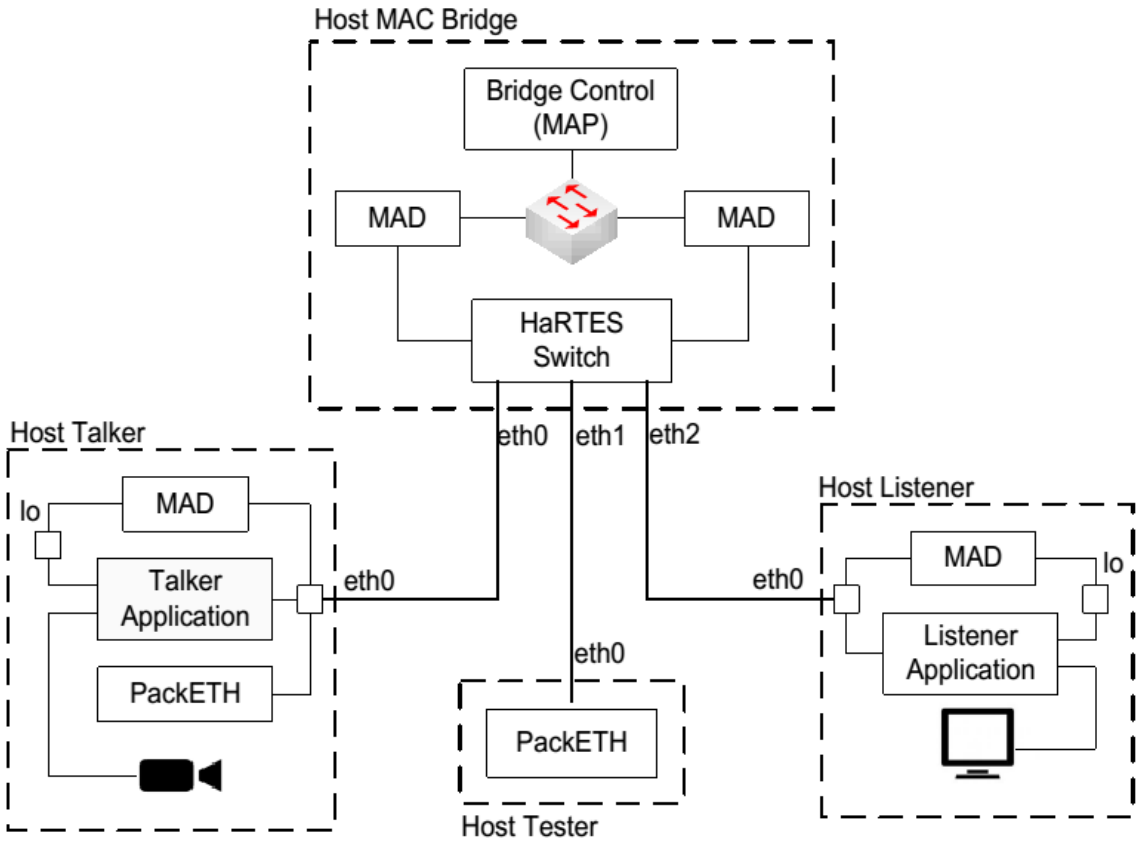

Figura 5.3: Estrutura provisória do HaRTES com suporte básico MSRP [32]

Com esta composição, à chegada de uma declaração *TalkerAdvertise* numa das portas, a camada de *switching* do HaRTES recebe esta mensagem do tipo MSRP e a reencaminha para o *MRPDaemon* da porta correspondente. No *MRPDaemon* é descodificada a mensagem, armazenando os seus atributos, e posteriormente reencaminhando-os para a *MRPBridge*. Neste módulo para além de serem armazenadas as características da *stream* desejada pela declaração, também é descodificado o seu tipo para de seguida, no caso do *TalkerAdvertise*, ser realizado o *broadcast* para os restantes *MRPDaemons* do *switch* para estes procederem ao envio da declaração para as estações terminais. Nesta etapa da arquitetura não foi ainda concebido um sistema que realizasse o controlo de admissão de pedidos de reserva, resultando na aceitação incondicional dos

pedidos e a sua propagação. Dentro da *MRPBridge* fica então registado que existe um *Talker* a querer comunicar a *stream* com o *StreamID* desejado, mas que ainda não se encontra presente um *Listener* para esta comunicação. À chegada da declaração ao *Listener*, este irá identificar-se como o seu destino imitindo um *ListenerReady* em direção à porta do *switch* que se encontra conectado. Novamente esta mensagem MRP é reencaminhada através do *MRPDaemon* desta porta até chegar à *MRPBridge*. Aqui é agora atualizado o estado da *stream* com a presença de um *Listener* para este *StreamID*. Também nesta etapa do desenvolvimento não existe qualquer reserva de recursos nem controlo de admissão, resultando novamente na propagação incondicional da declaração em direção ao *Talker* através do *MRPDaemon* correspondente à sua porta. O *Talker* é agora responsável por enviar a sua *stream*, que irá ser capturada na camada de *switching* do HaRTES e comunicada em *broadcast* por todas as restantes portas, não havendo qualquer encaminhamento especificado para a porta do *switch* do destino, nem o escalonamento do tráfego. O *switch* HaRTES irá apenas colocar todos os pacotes que recebe nas filas de tráfego assíncrono das restantes portas, apenas despachando estes quando existe oportunidade durante a janela assíncrona de comunicação, muito à semelhança do servidor de background previamente explicado. O sistema também é capaz de desfazer a instância da *stream* no MRPBridge, através de um pedido por parte de uma estação terminal.

5.2 Análise da Implementação Anterior

Nos subcapítulos anteriores tentou-se explicitar o estado prévio do projeto de integração do protocolo MSRP no *switch* HaRTES. Sendo assim é importante relembrar as componentes necessárias para apresentar uma arquitetura de rede capaz de fornecer aos seus nós garantias de qualidade serviço, e realizar uma

comparação com o estado do sistema previamente ao trabalho realizado durante esta dissertação. Estas componentes são:

- **Especificação do fluxo**: Embora as estações terminais sejam os únicos elementos da rede encarregues de especificar o fluxo de dados, foi desenvolvido, e encontra-se completo, a capacidade de descodificação e armazenamento desta informação no *switch* HaRTES, especificadas nas MSRPDUs emitidas pelas estações terminais.

- **Encaminhamento**: O mecanismo de encaminhamento segundo o protocolo MSRP encontra-se totalmente desenvolvido no que toca às mensagens protocolares que pretendem estabelecer as reservas de recursos. Em relação ao encaminhamento dos pacotes das próprias *streams*, a implementação necessitava de uma otimização, pois qualquer pacote que não seja do tipo FTT o *switch* HaRTES irá encaminhá-lo em *broadcast* por todas as suas portas. Isto gera um problema, pois acaba por preencher as filas de tráfego assíncrono de todas as portas do *switch*, exceto a que está associada à proveniência do pacote, gerando muita informação desnecessária que poderia ser destinada apenas a um *Listener*. No âmbito da presente dissertação foi implementado um mecanismo otimizado para o encaminhamento de pacotes relativos à própria *stream*.

- **Reserva de recursos**: O mecanismo para reservar recursos físicos no *switch* HaRTES para as *streams* descritas nas mensagens protocolares do MSRP, não existia, pelo que foi desenvolvido no âmbito da presente dissertação.

- **Controlo de admissão**: O mecanismo de controlo de admissão também não existia. Não era realizado qualquer controlo a nível dos pedidos de reserva, nem controlo do tráfego produzido por qualquer estação terminal MSRP.

- **Escalonamento de Pacotes**: O mecanismo de escalonamento de pacotes encontra-se completamente desenvolvido para o FTT-SE. No entanto para o tráfego proveniente de uma reserva MSRP isto não se evidencia, pois todos os

pacotes das *streams* emitidas pelas estações terminais do tipo MSRP eram colocadas numa fila que estava destinada a funcionar como servidor de background, prestando assim um serviço do tipo *best-effort*. Aliando as técnicas de escalonamento de tráfego do FTT-SE e um mecanismo para a reserva de recursos, pretende-se nesta dissertação desenvolver uma adaptação do sistema para conseguir escalonar os pacotes da *stream* com garantias de qualidade de serviço.

É possível verificar-se que tem vindo a ser realizado esforços para dotar o *switch* HaRTES de suporte para reserva de recursos MSRP. No entanto este ainda carece de mais desenvolvimento para poder prestar as garantias de qualidade de serviço que presta segundo o seu próprio protocolo. No âmbito desta dissertação foi então realizado todo o trabalho necessário para que o *switch* HaRTES passe a possuir todos os mecanismos necessários para ser compatível e responder de acordo com o paradigma FTT aos pedidos de reserva segundo o MSRP.

5.3 Plataforma de Desenvolvimento

Todo o trabalho descrito no ponto anterior foi realizado sobre quatro estações em Linux seguindo a linguagem de programação C, que até à data, tem vindo a ser utilizada no projeto para desenvolver o *switch* em *software*. Foi necessário que o ponto de partida do trabalho fosse uma versão do *switch* HaRTES estável e completamente despido de qualquer funcionalidade característica do MSRP, devido a alguns problemas de compatibilidade com os módulos *MRPBridge* e *MRPDaemon*. Desta forma a versão da arquitetura do *switch* HaRTES utilizada para iniciar o trabalho, encontra-se exibido na figura 5.4.

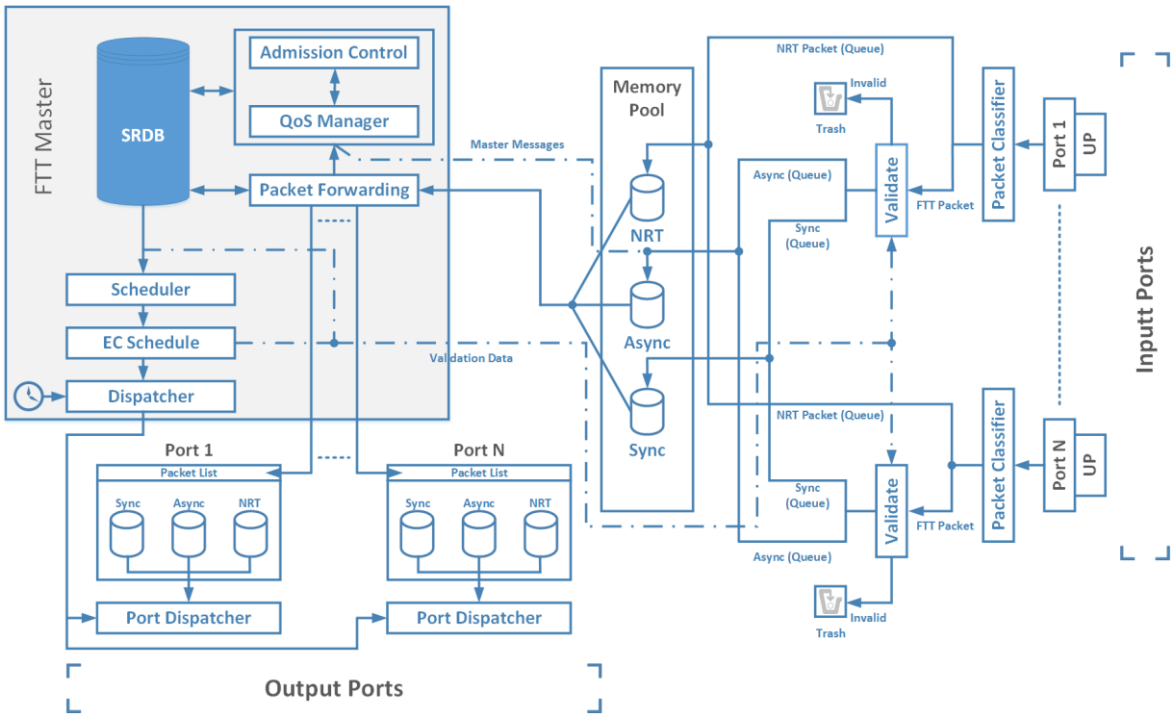

Figura 5.4: Estrutura base do switch HaRTES FTT-SE [33]

Com esta arquitetura, o HaRTES encontra-se capaz de fornecer garantias de qualidade de serviço a nós compatíveis com o FTT-SE, e fornecer um serviço do tipo *best-effort* a nós *standard*. Nesta arquitetura os pacotes dão entrada pelas *Input Ports*, e são de seguida classificados segundo o seu tipo (*FTT Packet* ou *Non-Real-Time Packet*). No caso de um pacote NRT este é colocado nas filas de memória internas do sistema reservadas para o seu tipo. No caso de um pacote FTT este é validado, podendo ser descartado, ou colocado numa das duas filas de memória interna reservadas para pacotes de tráfego síncrono ou assíncrono, segundo o seu tipo, finalizando assim o seu processo de receção de tráfego. Posteriormente o FTT *Master* contido no *switch* HaRTES fica encarregue de realocar os pacotes FTT, presentes nas filas memória internas, nas filas de memórias do tráfego indicado na porta de destino pretendida pelo pacote. É necessário relembrar que segundo o paradigma do FTT estes pacotes também passam por um controlo de admissão para verificar questões de *QoS*. O FTT Master é também responsável por escalonar toda a saída de tráfego que existe nas suas portas, segundo o paradigma FTT que estabelece a transmissão de dados organizada em ciclos elementares, com a

divisão destes em janelas temporais dedicadas ao tráfego síncrono e assíncrono já mencionado anteriormente.

Para facilitar a introdução e desenvolvimento das funcionalidades pretendidas todo o trabalhado realizado foi implementado apenas com a camada de *switching* do HaRTES ativa, não existindo o escalonamento de pacotes FTT, mas cumprindo as janelas temporais para a transmissão de pacotes segundo o seu paradigma.

É necessário também mencionar que o *switch* HaRTES foi emulado numa plataforma em Linux com as capacidades necessárias a desenvolver a aplicação e detentor de suficientes *NICs* (ou portas) de *Ethernet* para este funcionar como um *switch*. As estações terminais da rede foram emuladas num computador pessoal detentor do *Linux* e mais duas plataformas em *Linux*. Nas estações terminais foram desenvolvidas pequenas aplicações com a ajuda do *MRPDaemon* para estabelecer qualquer teste segundo o MSRP.

5.4 Trabalho Desenvolvido

O objetivo de todas as atividades aqui apresentadas será fornecer uma resposta às necessidades presentes no *switch* HaRTES em relação ao MSRP. Dentro destas atividades apresentadas encontram-se a diferenciação de tráfego, a criação da *MAC Address Fowarding Table* e a distribuição de pacotes segundo reservas, que pretendem responder às necessidades de encaminhamento encontradas. Será também exposto os servidores planos, criados com objetivo adicionar ao *switch* HaRTES estruturas físicas indicadas para a reserva de recursos, considerando tanto o MSRP como o paradigma FTT. Será apresentado como uma solução para a problemática do controlo de admissão, a implementação do MAP baseado na estrutura do FTT e o algoritmo de decisão que rege o funcionamento deste. Dentro das atividades também está a criação de estruturas que relacionem o FTT com os

pedidos de MSRP, a alteração da janela do EC que rege o envio de tráfego síncrono e assíncrono pelas portas do *switch* e o *upgrade* realizado no *port dispatcher* para poder utilizar os servidores como resposta às necessidades relativas ao escalonamento de pacotes.

5.4.1 Gestão dos Pacotes MSRP

Como já foi mencionado, antecedente a este trabalho existiu a criação do *MRPDaemon* e *MRPBridge*, no entanto foi necessário melhorar a integração destes no *switch* HaRTES. Utilizando o mesmo mecanismo de conexão de cada porta do *switch* com o seu próprio *MRPDaemon* através de redes virtuais, foi realizado uma filtragem de pacotes de MSRP, graças ao seu identificador próprio no pacote de Ethernet. Assim sendo, a chegada de um pacote por uma das suas portas de *input*, ativa a *thread* do evento de receção (*rx_event_handler*) na camada de *switching* correspondente a esta porta, e aqui é verificado se o pacote em questão possui o identificador MSRP. Caso este possua o identificador, a *thread* do evento realiza o encaminhamento do pacote para o *MRPDaemon* associado a esta porta e fica à espera de um próximo pacote, como se encontrava antes de ser ativada. Caso não possua, o *switch* resume os procedimentos normais característicos na receção de pacotes pelas suas portas.

Também seguindo o mecanismo anteriormente implementado, todos os *MRPDaemons* encontram-se conectados a uma só *MRPBridge*, que está encarregada de fazer o encaminhamento dos pacotes MSRP. Aqui foi concebido um meio de comunicação entre a *MRPBridge* e a camada de *switching* do *HaRTES*. Para isto foram utilizados mecanismos que possibilitam a leitura e escrita de uma zona de memória partilhada entre este módulo e a *thread mrp_control_event*, na camada de *switching*, criada para atender eventos gerados pela *MRPBridge* quando esta necessita de realizar uma decisão característica do MAP. Para assegurar a

sincronização dos dois processos, foram implementados dois semáforos que regem os acessos à memória partilhada, para que não existam dois acessos simultâneos que poderiam provocar a corrupção da informação trocada.

Nesta zona de memória partilhada foram criadas variáveis correspondentes à informação característica das *streams* contidas no campo *FirstValue*, das MSRPDUs, e às variáveis que ditam pedidos de ações a realizar e as suas respostas entre os dois pontos de acesso. Desta forma a *MRPBridge* consegue assinalar à camada de *switching* que quer proceder a uma reserva de recursos, através da variável de controlo *cmd_bridge*. A *MRPBridge* aguarda uma resposta vinda da *thread mrp_control_event* que lhe indica, através da variável de controlo *cmd_hartes*, se é possível ou não esta reserva. O mesmo foi realizado para destruição de reservas, sendo importante de mencionar que os únicos parâmetros de controlo MSRP que são armazenados inalterados no próprio HaRTES são o *StreamID*, que contém o endereço de MAC do *Talker* e o identificador único da *stream*, o endereço MAC de destino do *Listener* e a prioridade da *stream*.

Assim sendo, é possível estabelecer uma separação entre todos os módulos referentes ao MSRP, sendo que toda a atividade pertinente ao protocolo se realiza diretamente dentro destes, e tudo o que se encontra dentro domínio do HaRTES, como por exemplo o estado dos servidores ou recursos disponíveis e a sua reserva, é tratado na camada de *switching*. Na figura 5.5 é apresentada uma ilustração desta arquitetura para um *switch* de duas portas, com todas as componentes e aplicações relevantes para esta dissertação.

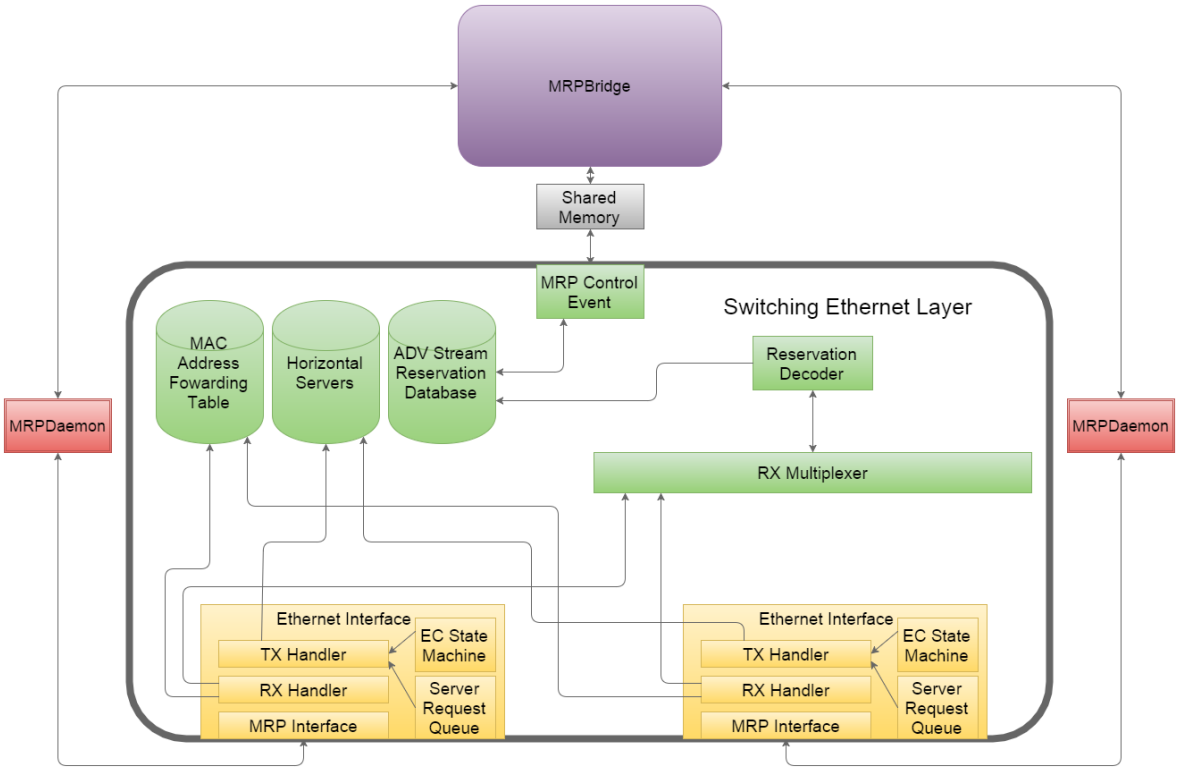

Figura 5.5: Estrutura desenvolvida para um Switch HaRTES com duas portas

5.4.2 MAC Address Fowarding Table

O *switch* HaRTES possui mecanismos próprios para lidar com os endereços MAC a que as suas portas acedem. No entanto como apenas a camada de *switching* de Ethernet se encontra ativada durante o desenvolvimento do trabalho realizado nesta dissertação, foi necessário para aplicações que serão descritas nas secções seguintes, desenvolver um mecanismo que conseguisse manter todos os endereços MAC a que uma porta do *switch* consegue aceder.

Para isto, foi criado uma *Port Table* que é instanciada ao início do sistema, de acordo com o número de portos que o *switch* possui. É possível através desta tabela aceder a todas as propriedades de cada porto, tais como: o identificador do número do porto, informações sobre as interfaces, as filas de envio e receção de todos os tipos de pacotes. As funcionalidades que foram aqui também introduzidas e que têm ligação às secções seguintes são: o identificador da lista de

servidores horizontais do porto, as filas de pedidos de reserva e os endereços MAC alcançados pelo porto.

Para preencher esta *Port Table*, no evento de receção de pacotes em cada um dos portos, é filtrado o endereço MAC de origem do pacote e, caso este ainda não se encontre presente na entrada da *Port Table* correspondente ao seu porto, é armazenado. Isto acontece tanto para pacotes protocolares do MSRP como qualquer outro tipo de pacotes.

5.4.3 Adaptação de Estruturas FTT com Suporte para o MSRP

O *switch* HaRTES possui estruturas muito específicas para os requisitos de cada mensagem FTT. Apesar de querermos desenvolver suporte para o MSRP no HaRTES, não queremos alterar qualquer tipo de componente essencial para as principais estruturas temporais já montadas ou provocar grandes mudanças no domínio do funcionamento do FTT. Com esse intuito, foi replicado a principal tabela utilizada nos requisitos das mensagens do FTT. Os seus principais campos são:

- **m_id**: correspondente ao identificador da mensagem;

- **m_size**: correspondente ao tamanho da mensagem em bytes;

- **m_max_size**: correspondente ao tamanho máximo da mensagem em bytes;

- **m_MTU**: correspondente à unidade de transmissão máxima;

- **m_period**: correspondente ao período da mensagem em número de ECs que tem tradução direta da variável do MSRP *MaxIntervalRate*;

- m_deadline: correspondente à *deadline* da mensagem em número de ECs;

- **m_init**: correspondente ao EC onde existe a primeira ocorrência;

- **m_timeliness**: indica se o tipo de mensagem é síncrono ou assíncrono;

- **sw_consumer_idx**: lista dos identificadores dos *slaves* consumidores de tráfego;

- **sw_producer_idx**: lista dos produtores dos *slaves* produtores de tráfego;

Como mencionado esta estrutura de tabela foi preservada e replicada para ser gerida pela *thread mrp_control_event* e uma duplicação desta tabela também existe com a adição de novos parâmetros para fazer a correspondência ao MSRP, e por ventura também ajudar a fazer a correlação de informação entre a *MRPBridge* e o HaRTES. Os novos campos adicionados são:

- **StreamID**: correspondente à informação contida no campo *FirstValue* identificando a *stream*;

- **state**: correspondente ao estado da *stream*;

- **dest_Addr**: correspondente ao endereço MAC do *Listener* de destino da *stream*;

- **src_Addr**: correspondente ao endereço MAC do *Talker* de origem da *stream*;

- **C**: correspondente ao tamanho máximo do pacote em bytes que tem tradução direta da variável do MSRP *MaxFrameSize*;

- ***priority***: correspondente à prioridade da *stream*;

- ***ServerID***: identificador correspondente à fila de pacotes na porta de destino do *switch* que funciona como servidor para esta *stream*;

Esta nova estrutura é identificada por *ADV_MESSAGE_REQUESITS*, e irá ser gerida, tal como a original de onde foi duplicada, pela *thread mrp_control_event* segundo os comandos ditados pela *MRPBridge*, para os eventuais pedidos para realizar ou desfazer a reserva de recursos. Com isto mantem-se um certo grau de

compatibilidade entre o FTT e o MSRP para que, em implementações futuras, qualquer cálculo realizado pelos módulos FTT tenham em conta as características dos pacotes contidos nas *streams* provenientes das estações terminais com MSRP, ao mesmo tempo que se mantém um certo isolamento entre os mecanismos dos dois. Também convém mencionar que foi desenvolvido todo um conjunto de funções extras para lidar com esta nova tabela (*adv_SRDB*) com vista a ajudar na resolução de problemas que serão discutidos nas secções seguintes.

5.4.4 Servidores Planos

Para fornecer ao *switch* HaRTES recursos físicos que possam suportar os pedidos de reserva das *streams*, e manter a estrutura de funcionamento base do *switch* segundo a arquitetura FTT, foi desenvolvido um módulo capaz de oferecer a criação, destruição e manuseamento de servidores a cada uma das portas do HaRTES. O funcionamento base deste módulo apresenta um *background server* para cada uma das portas do *switch*, apresentado a possibilidade de criação de múltiplos *deferrable servers*.

A arquitetura deste módulo, ilustrado na figura 5.6, consiste numa tabela de servidores (*ServerTable*), em que cada entrada desta tabela (*ServerTableItem*) será inicializada durante a instanciação das componentes de cada porta, resultando assim num número igual de entradas e número de portas do *switch*. Por uma questão de concepção cada uma destas entradas terá acesso imediato ao servidor de maior prioridade (*HP_server*), sendo por defeito após a instanciação deste módulo um servidor de background que não possui qualquer tipo de controlo de capacidade.

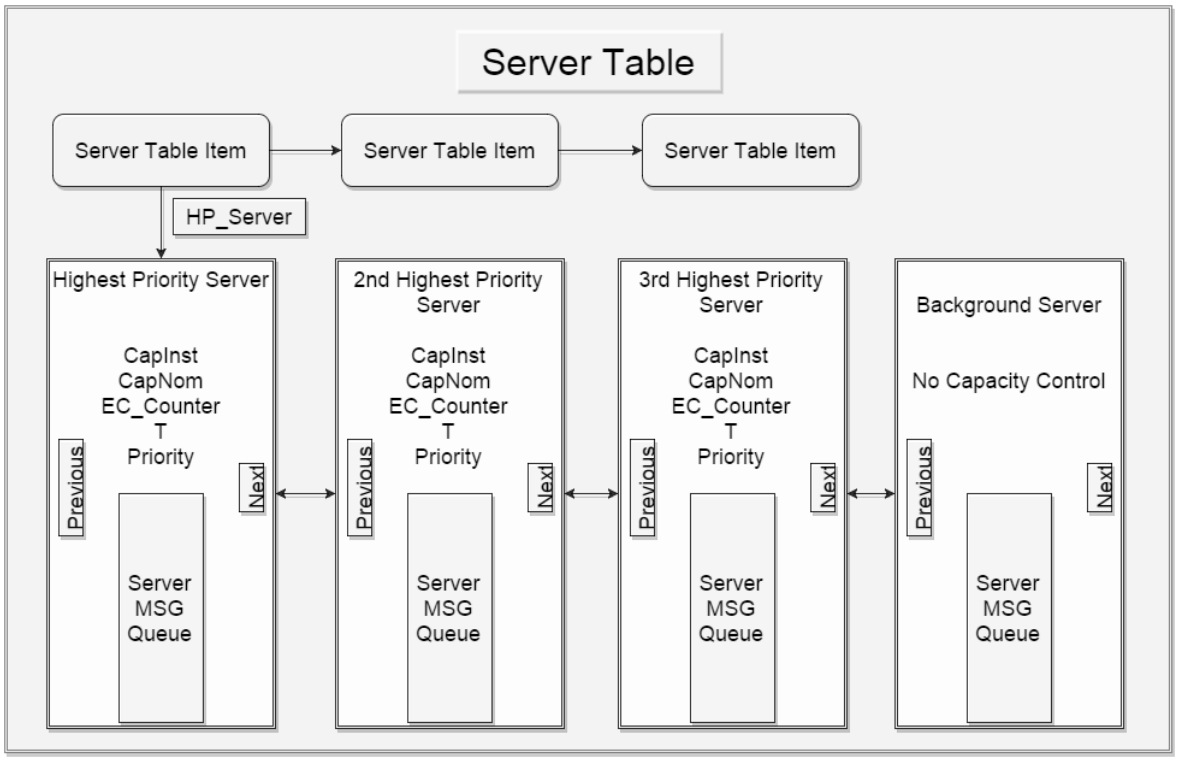

Figura 5.6: Estrutura do módulo dos Servidores

À medida que for necessário a criação de novos servidores, todos estes do tipo *deferrable*, será possível instanciar um novo servidor (*ServerData*) com as seguintes características:

- **CapInst**: corresponde à capacidade instantânea do servidor em bytes sendo inicializada com o mesmo valor da capacidade nominal. Quando é realizado um *dispatch* de um pacote contido num servidor, é subtraído a esta variável o tamanho do pacote;

- **CapNom**: corresponde à capacidade nominal do servidor em bytes. Este valor é estático e corresponde ao número bytes possível de ser enviado durante o período deste;

- **EC_counter**: variável de contagem que possibilita a medição temporal e consequentemente o refrescamento da capacidade instantânea do servidor quando este valor iguala o valor presente em T;

- **T**: variável temporal estática medida em número de ECs correspondente ao período dos instantes de refrescamento da capacidade instantânea;

- **Priority**: variável estática correspondente à prioridade da *stream* e por consequência à prioridade do servidor.

- ***ServerMsgQueue*:** estrutura de filas ligadas que possibilita o armazenamento dos pacotes de Ethernet a serem enviados pelas portas do *switch*. Esta fila é gerida segundo a política *First Come First Served*.

A criação de um novo servidor pode ser realizado passando como argumentos à função *AddServer* o identificador do *ServerTableItem* correspondente à porta desejada, o *ServerID* desejado, a prioridade do servidor, a capacidade nominal e o período de refrescamento em ECs deste. O módulo irá posicionar o novo servidor dentro da lista ligada de *ServerData* segundo a sua prioridade em relação aos demais, garantindo o acesso ao servidor de maior prioridade sem ter de percorrer as entradas de servidores de menor prioridade. Caso sejam encontrados servidores com a mesma prioridade, estes também são ordenados por ordem de chegada, dos mais antigos para os mais recentes. A destruição de um servidor pode ser realizado passando como argumentos à função *RemoveServer* o identificador do *ServerTableItem* corresponde à porta desejada e o *ServerID* do servidor desejado.

No *switch* HaRTES existe uma *Ethernet Memory Pool* dedicada a pacotes de *Ethernet*, onde os pacotes são colocados à medida que chegam, e retirados à medida que saem. Entre as diferentes filas de tráfego presentes no HaRTES e esta *memory pool* é trocada apenas o endereço de memória necessária para aceder ao pacote. Mantendo esta estrutura de operações foram concebidas funções para armazenar e retirar pacotes dos servidores. Para colocar pacotes Ethernet nos servidores, é disponibilizado a função *HorizontalServer_PutMsg* que espera como argumentos o identificador do *ServerTableItem* corresponde à porta desejada, o

ServerID do servidor desejado, o tamanho do pacote e o ponteiro para a zona de memória do pacote.

Para obter pacotes de *Ethernet* dos servidores, é disponibilizado a função *HorizontalServer_Dispatch* que espera como argumentos apenas o identificador do *ServerTableItem* corresponde à porta desejada. Esta função *HorizontalServer_Dispatch* foi concebida para responder às necessidades de um *dispatcher* para tráfego assíncrono, que fará conexão com a camada de *switching* descrita numa secção posterior. Aqui serão percorridos todos os servidores correspondentes a esta porta, dos mais prioritários até aos menos, até encontrar um que possua um pacote para ser despachado. É tida em conta também a capacidade instantânea do servidor no momento do *dispatch*, impossibilitando assim a transmissão de pacotes caso se verifique que o tamanho do pacote a despachar não cabe na capacidade disponível pelo servidor. Dado este acontecimento, a função de *dispatch* continua a percorrer os servidores até achar um pacote que cumpra este requisito. É de notar então que todo o tráfego assíncrono que é colocado no servidor de background só é atendido quando não existe mais nenhum pacote disponível no momento do *dispatch*.

Para realizar a tarefa de refrescamento das capacidades dos servidores foi concebido a função *RefreshCapacity* que recebe como argumentos o identificador do *ServerTableItem*. Isto é devido ao facto de cada uma das portas do *switch* possuir a sua própria *thread* que gere as janelas temporais do EC. Desta forma, no final de um ciclo esta função é chamada e percorre a lista ligada de servidores da porta correspondente, incrementando as variáveis *EC_counter* caso estas sejam inferiores ao seu T correspondente e reinicializando aquelas que igualem o seu T, procedendo atualização do seu *CapInst* com o valor decretado na variável *CapNom*.

5.4.5 Instanciação de reservas

Apesar de ser possível criar servidores a qualquer instante, esta liberdade pode causar danos a nível da qualidade de serviço apresentada. Para combater isto, em vez de se realizar uma reserva mal se verifique que existem garantias de qualidade de serviço para oferecer à *stream*, é criado um pedido de reserva interno que só será atendido durante a janela temporal da TM.

Desta forma é então instanciado para cada porta uma fila de pedidos aos servidores (*server_request_queue*), que terá como objetivo armazenar todos os pedidos a realizar, quer sejam estes a destruição de um servidor específico ou a criação de um servidor segundo um certo conjunto de características.

Esta fila é preenchida na *thread mrp_control_event*, sempre que aceite um pedido de reservas ou destruição destas. É de relembrar que esta *thread* possui uma tabela, já antes mencionada no capítulo 5.4.3, com o estado de todas as reservas de recursos para as *streams* no sistema.

As ações a realizar contidas nesta fila de pedidos aos servidores, são executadas na *thread* de eventos de envio de pacotes (*tx_queue_handler*) que é regida pela *EC_state_machine* que regula os instantes das mudanças das janelas temporais. Isto assegura que estes pedidos só são realizados durante a janela temporal da TM.

5.4.6 Diferenciação e Distribuição de Tráfego

O *switch* HaRTES possui mecanismos próprios para a diferenciação de tráfego FTT, realizando o seu encaminhamento segundo a porta de destino e tipo de fila, síncrona ou assíncrona, a colocar. No entanto qualquer tipo de tráfego que não seja FTT é replicado em todas as filas de NRT das restantes portas. Com a introdução dos servidores é agora possível criar várias filas diferentes, cada uma

por exemplo para uma *stream* de dados específica. No entanto mesmo possuindo servidores é ainda necessário o uso da *MAC Address Fowarding Table* e da tabela com o estado de todas as reservas no sistema, mencionada no capítulo 5.4.3, para proceder a correta diferenciação e distribuição de tráfego.

À chegada de pacotes na porta de entrada do *switch*, que não sejam do tipo MSRP, as *threads* de eventos de receção colocam os pacotes nas filas memória internas do sistema. Estas filas são depois geridas por uma única *thread* (*rx_multiplexer*) que realiza o encaminhamento dos pacotes aqui armazenados para as respetivas filas e portas de saída. Sendo que não existe qualquer tipo de mecanismo que faça o correto encaminhamento do tráfego das *streams* reservadas segundo o MSRP, foram introduzidas nesta *thread* novas funcionalidades para realizar a correspondência do pacote da *stream* com a própria entrada na tabela do estado de reservas no sistema, e posteriormente com a porta de saída e servidor correto.

Para isto foi desenvolvido todo um conjunto de operações que permite rastrear o endereço MAC de origem e destino do pacote, comparar estes endereços com os endereços presentes na tabela com o estado das reservas, resultando numa possível correspondência destes dois campos capaz de nos fornecer com o identificador do servidor associado a esta reserva de recursos. Caso não exista uma correspondência, significando que não existe nenhuma reserva para este conjunto de endereços MAC, o pacote é distribuído por todas as portas do *switch*, exceto a porta por onde este pacote entrou, e colocado no servidor de background das portas. No caso de uma correspondência o pacote em questão é colocado no servidor reservado para a sua *stream*, correspondente ao identificador encontrado, na porta de saída desejada, especificada graças à análise do endereço MAC de destino e das funcionalidades desenvolvidas para manusear a *MAC Address Fowarding Table*. A passagem do endereço do pacote para o servidor desejado é realizado através da função *HorizontalServer_PutMsg* já explicada anteriormente.

5.4.7 Dispatcher

O despacho dos pacotes contidos dentro da memória do *switch* HaRTES para meio físico da rede acontece nas *threads* de eventos de envio (*tx_queue_handler*). Esta *thread* é gerida como uma máquina de estados, regulada pelas janelas temporais que compõem o EC característico do FTT. Assim sendo a *thread* está regida por quatro estados: *TM Window*, *Sync Window*, *Async Window* e *Idle Window.*

Todo o tráfego das *streams* reservadas segundo o MSRP é tratado como tráfego assíncrono e armazenado nos servidores criados para estas reservas. Nesta *thread* foi então só necessário realizar a chamada da função *HorizontalServer_Dispatch*, com o identificador que relaciona a porta com a sua lista de servidores, para encontrar o servidor mais prioritário que possui pacotes para enviar e apresenta a capacidade necessária para o envio deste. A *thread* chamará esta função enquanto se encontrar na janela de comunicação assíncrona da respetiva porta. É necessário frisar então que o pacote selecionado por esta função resulta de um escalonamento baseado no algoritmo do *Deferrable Server*, anteriormente explicado.

5.4.8 MAP – Baseado na estrutura do FTT

Após o desenvolvimento de todas as aplicações para realizar a comunicação entre módulos, mecanismos de encaminhamento, reserva de recursos, gestão de servidores e escalonamento de pacotes, foi possível nesta etapa de desenvolvimento ter uma plataforma em que apenas falta o controlo de admissão de pedidos de reservas. Este controlo irá garantir que o sistema só aceitará pedidos se possuir recursos necessários para prestar a qualidade de serviço

desejada por uma *stream*. Apesar de haver algoritmos avançados para o cálculo do tempo de resposta de mensagens para o protocolo FTT-SE [34], por falta de tempo optou-se usar nesta dissertação uma adaptação simples da análise clássica de tempo de resposta para sistemas de prioridades fixas não preemptivos, que se apresenta de seguida.

Para entender este cálculo é preciso explicitar que na transmissão de uma mensagem por parte do *switch* HaRTES esta pode estar sujeita a bloqueios, devido à transmissão de mensagens de menor prioridade no mesmo meio, e interferências, devido à transmissão de mensagens de maior prioridade no mesmo meio. É preciso considerar que tais bloqueios e interferências podem acontecer durante o *uplink*, comunicação entre o produtor da mensagem e o *switch*, e o *downlink*, comunicação entre o *switch* e o recetor da mensagem, como mostrado na figura 5.7. Adicionalmente é preciso considerar também que o próprio *switch* apresenta uma latência, característica do processo de encaminhamento da mensagem entre as suas portas.

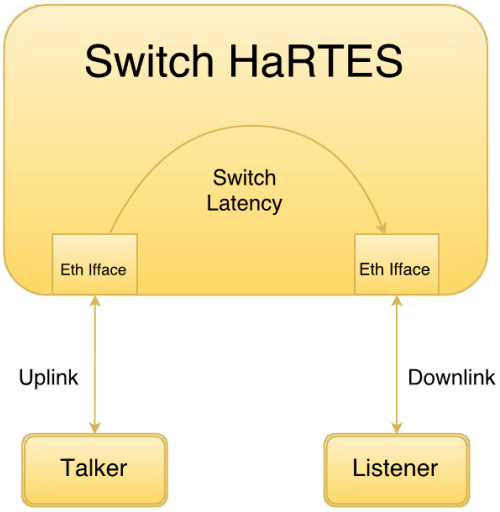

Figura 5.7: Topologia de um Switch HaRTES de duas portas (adaptado de *Ashjaei et al.* [34])

Assim sendo, será necessário calcular o tempo de resposta de uma mensagem assíncrona, caracterizada no pedido de reserva da *stream* MSRP,

transmitida do *Talker* para o seu *Listener*, calculando inicialmente o tempo de resposta no *uplink* seguido do cálculo do tempo de resposta no *downlink*. Por fim somando estes dois tempos à latência do *switch* teremos o tempo de resposta total para a transmissão de uma mensagem desde o *Talker* até ao *Listener*. Devido ao tempo de latência estar diretamente relacionado com todos os processos de encaminhamento de pacotes, foi estudado um caso onde o *switch* possui 25 servidores diferentes na porta de destino da mensagem, sendo o servidor desejado para o encaminhamento o último deste conjunto. Para encaminhar o pacote o *switch* precisa de percorrer todos os servidores até chegar ao desejado. Com este caso pretende-se assim obter um tempo de latência fixo que será utilizado no cálculo do tempo de resposta total da mensagem.

Podemos então apresentar que o *worst* case response time (WCRT) será calculado através da seguinte soma dos seguintes tempos:

$$WCRT_{Total} = WCRT_{UP} + T_{Switch} + WCRT_{Down}$$

Sendo:

$WCRT_{UP}$ – Tempo de resposta no *uplink*;

T_{Switch} – Tempo de latência no *switch*, determinado através do caso de estudo;

$WCRT_{Down}$ – Tempo de resposta no *downlink*;

Será necessário agora apresentar o método para calcular os tempos de resposta que são utilizados no cálculo do tempo de resposta total [34]:

$$WCRT = \frac{C_i}{\alpha_i} + I_i + B_i$$

Sendo:

C_i – Tempo total de transmissão da mensagem m_i;

α_i – Fator de inflação, calculado apenas no downlink e igualado a 1 no uplink;

I_i – Tempo de atraso causado pela interferência de mensagens de maior prioridade;

B_i – Tempo de atraso causado pelo bloqueio por parte de mensagens de menor prioridade;

O fator de inflação, utilizado apenas no *downlink* visto que os *Talkers* que reservam recursos segundo o MSRP não possuem o paradigma FTT para envio de mensagens, será dado por:

$$\alpha_i = \frac{\min(LW - Id)}{EC}$$

Sendo:

LW – Duração da janela assíncrona;

Id – *Idle time*, correspondente ao tamanho máximo do pacote das mensagens de maior ou igual prioridade;

EC – Duração do ciclo elementar;

O tempo de atraso que caracteriza o bloqueio por parte de mensagens de menor prioridade será dado pelo valor máximo do tempo total de transmissão das mensagens de menor prioridade:

$$B_i = \max_{j \in lpe(i)} C_j$$

Sendo:

C_j – Tempo total de transmissão de uma mensagem m_j, que possui uma prioridade inferior por comparação a m_i;

O atraso causado por mensagens de maior prioridade é obtido por um processo iterativo que converge num número finito de passos. No caso de este

tempo calculado ultrapassar o valor do período da mensagem, esta mensagem não será possível de ser escalonada com o resto das mensagens presentes. O processo pode ser descrito por:

$$I_i(0) = \sum_{k \in hp(i)} C_k$$

$$I_i(m) = \sum_{k \in hp(i)} \left\lceil \frac{I_i(m-1)}{T_k} \right\rceil * C_k$$

Sendo:

$C_k -$ Tempo total de transmissão de uma mensagem m_k, que possui uma prioridade superior por comparação a m_i;

$T_k -$ Período da mensagem m_k, neste caso representa o mit pois é aplicado a mensagens assíncronas;

O cálculo destes tempos é disponibilizado como funcionalidade na aplicação que gere a tabela de reservas de recursos no sistema (*adv_SRDB*). Através da função *ADV_MESSAGES_TABLE_calculate_rwc* passando como argumentos as características da reserva de *stream* realizar, é possível determinar se o *WCRT_Total* ultrapassa ou não o valor do período da mensagem caracterizadora da *stream*. No caso de ultrapassar conclui-se então que com o conjunto de reservas agora instanciadas não é possível escalonar, com garantias de qualidade de serviço, as mensagens desta *stream*. Com isto a *thread mrp_control_event* informa a *MRPBridge* da impossibilidade de realizar reservas, para posterior comunicação do acontecimento às estações terminais intervenientes. Caso não ultrapasse o algoritmo garante então a qualidade de serviço com as reservas indicadas no pedido da *stream*. Com isto é instanciado então o pedido aos servidores da porta de destino, resultando na atual reserva de recursos graças à criação de um servidor para tráfego dedicado a esta *stream*.

Assim sendo ficam implementados todos os módulos, aplicações e funcionalidades concebidas com o intuito de fornecer ao *switch* HaRTES todas as capacidades imprescindíveis para a gestão da rede de tempo-real com reservas segundo a estrutura do MSRP.

6. Testes e Validação

Todas as interações entre módulos, estruturas e mecanismos implementados e descritos no capítulo anterior foram sujeitos a testes de validação. No presente capítulo são descritos todos os testes realizados considerados pertinentes e analisados os resultados obtidos assim como também a metodologia usada para os realizar.

6.1 Metodologia

No decorrer de todo o trabalho desenvolvido durante a dissertação foram desenvolvidos uma série de testes com o intuito de comprovar os mecanismos de encaminhamento, reserva de recursos, controlo de admissão, escalonamento de pacotes e um conjunto de procedimentos indispensáveis para o correto funcionamento das restantes funcionalidades.

Para realizar esses testes foram utilizados três computadores com o intuito de emularem estações terminais e um outro computador com o intuito de emular o *switch* HaRTES através de *software*. Na figura 6.1 encontra-se presente os processos que emulam o *switch* HaRTES em semelhança ao sistema apresentado na figura 5.5 só que aqui com três portas, é possível visualizar a consola inferior que representa a camada do HaRTES, as três consolas intermédias que representam os MRPDaemons das três portas do *switch*, e por fim a consola superior que representa a MRPBridge.

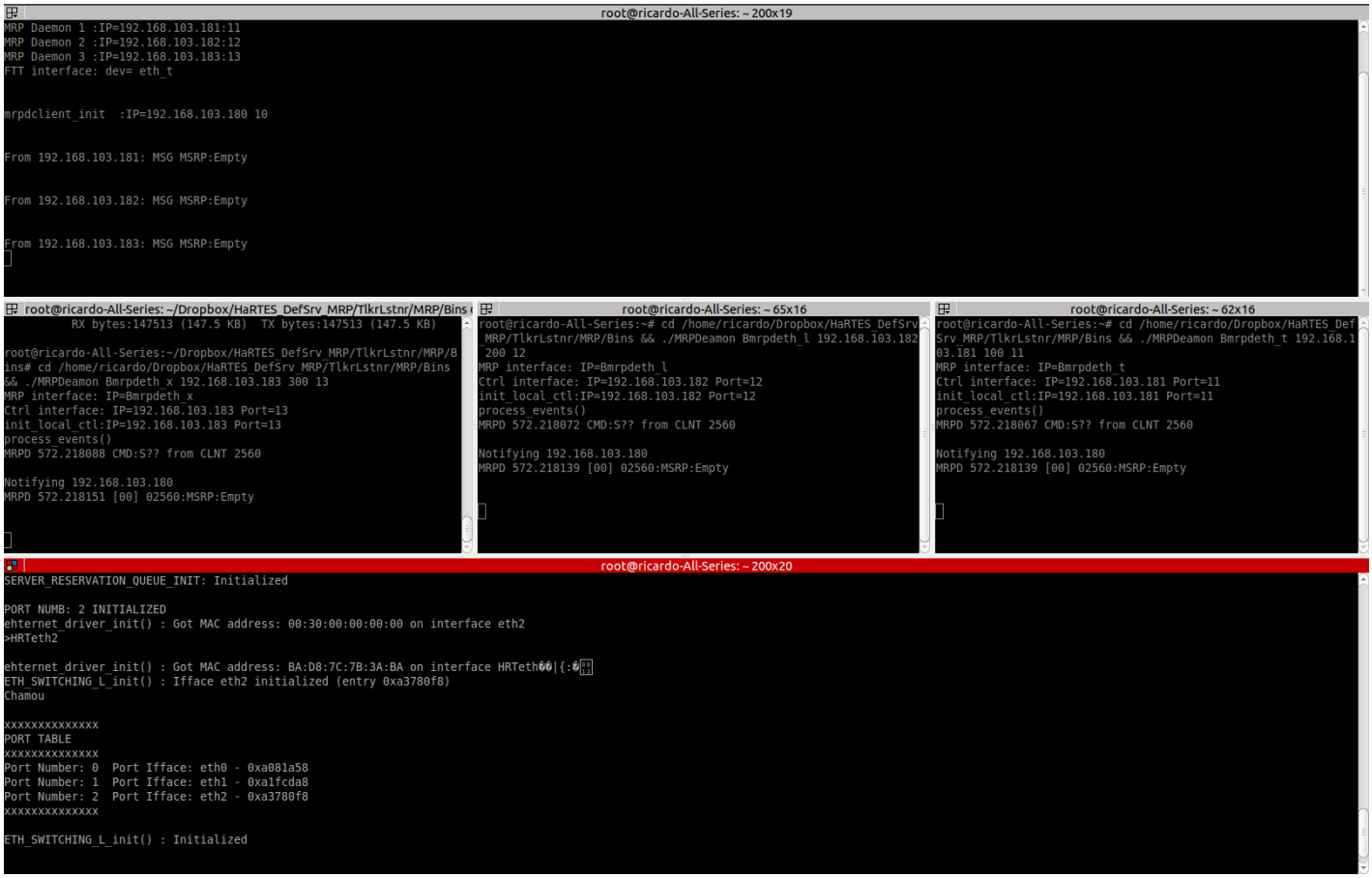

Figura 6.1: Processos que representam o *Switch* HaRTES de três portas

Para testar e validar todas as atividades do *switch* HaRTES para além de ser utilizada a interação através das consolas para sinalização de eventos internos, foram também utilizadas as ferramentas *PackEth* e *Wireshark* para gerar tráfego com características específicas e o analisar o percurso dos pacotes respetivamente. Desta forma os computadores que emulam as estações terminais através do uso das aplicações de testes, conseguem declarar pedidos de reservas e enviar-los para o computador que emula o *switch* HaRTES, e através do uso do PackEth conseguem gerar o tráfego com as características declaradas nos pedidos de reserva MSRP. Através do uso do *Wireshark* as estações terminais que funcionam em certos instantes como *Listeners* podem também listar os pacotes recebidos para posterior análise. No computador que emula o *switch* HaRTES apenas se fez uso das interações disponibilizadas através das consolas e do *Wireshark* para analisar o

encaminhamento dos pacotes de pedidos de reserva MSRP entre os módulos MRPDaemon, MRPBridge e a camada *Ethernet Switching* do HaRTES.

6.2 Especificação de Requisitos do Sistema

Para realizar um correto teste e validação é fundamental, previamente ao desenvolvimento do trabalho, realizar uma especificação das funcionalidades desejadas a implementar. Tendo como base as componentes necessárias para uma arquitetura de rede ser capaz de prestar garantias de qualidade de serviço aos seus nós, o sistema no início do trabalho desenvolvido apresentava carências a nível do encaminhamento, reserva de recursos, controlo de admissão e escalonamento de pacotes. Através destas componentes gerais foi possível delinear e conceber a arquitetura do sistema a implementar, de tal forma que encontram-se especificados na tabela 6.1 os requerimentos do sistema estipulados com intuito de garantir o cumprimento destas componentes.

Componente Referente	***Requisito***	***Descrição do requisito***
Especificação do Fluxo/ Encaminhamento	**Filtragem e gestão de pacotes de MSRP**	Filtragem e encaminhamento para o MRPDaemon desejado de forma a não serem propagadas pela rede através da camada de *Ethernet Switching* do HaRTES mensagens de controlo
Encaminhamento/ Controlo de Admissão	**Filtragem e gestão de endereços MAC**	Filtragem e armazenamento dos endereços MAC de origem dos pacotes de dados que são possíveis alcançar através de uma determinada porta

Especificação do Fluxo/ Encaminhamento/ Controlo de Admissão	**Compatibilidade e modularidade das estruturas e mecanismos FTT com suporte para MSRP**	Criação ou adaptação de estruturas capazes de armazenar o estado das reservas existentes no sistema e a suas características sem comprometer o funcionamento característico do *switch* HaRTES
Reserva de Recursos/ Escalonamento de Pacotes	**Capacidade de armazenamento de pacotes em servidores dedicados a reservas específicas**	Criação de estruturas denominadas por servidores com o intuito de armazenar, segundo as reservas instanciadas pelas mensagens de controlo MSRP, os pacotes de tráfego relativos a estas
Encaminhamento/ Controlo de Admissão	**Diferenciação dos pacotes de tráfego relativos a reservas**	Filtragem e encaminhamento dos pacotes para os servidores das devidas portas de saída segundo a existência de uma reserva
Reserva de Recursos/ Escalonamento de Pacotes	**Criação dos servidores fora das janelas temporais dedicadas para o envio de tráfego**	Criação de servidores correspondes aos pedidos de reserva somente no instante apropriado de modo a não comprometer o funcionamento do sistema durante o envio de pacotes
Controlo de Admissão	**Mecanismo de controlo de admissão de pedidos de reserva**	Mecanismo de controlo de aceitação de pedidos de reserva MSRP, segundo uma análise das características da *stream* e dos recursos disponíveis.

Escalonamento de Pacotes	**Despacho de tráfego assíncrono na janela correspondente e segundo um critério de prioridades e capacidade dos servidores**	Despacho dos pacotes de tráfego assíncrono que foram previamente armazenados nos servidores criados para as suas específicas *streams* de dados segundo as características decretadas nos pedidos de reserva MSRP

Tabela 6.1: Especificação dos requisitos do projeto

6.3 Teste e Validação dos Requisitos do Sistema

Após o delineamento dos requisitos do sistema a implementar procedeu-se ao desenvolvimento de todas as funcionalidades necessárias para o *switch* HaRTES se tornar totalmente autónomo e compatível com os mecanismos de reservas de recursos MSRP. Em anexo estão apresentadas um conjunto de tabelas que visam explicitar os testes realizados, pretendendo validar as funcionalidades necessárias para o cumprimento dos requisitos. Também é necessário mencionar que a sequência que se segue revela também a ordem pela qual as funcionalidades foram implementadas. A disposição do conteúdo das tabelas é o seguinte:

- Na tabela A.1 encontram-se os testes efetuados para o requisito **Filtragem e gestão de pacotes MSRP**;

- Na tabela A.2 encontram-se os testes efetuados para o requisito **Filtragem e gestão de endereços MAC**;

- Na tabela A.3 encontram-se os testes efetuados para o requisito **Compatibilidade e modularidade das estruturas e mecanismos FTT com suporte para MSRP**;

- Na tabela A.4 encontram-se os testes efetuados para o requisito **Capacidade de armazenamento de pacotes em servidores dedicados a reservas específicas;**

- Na tabela A.5 encontram-se os testes efetuados para o requisito **Diferenciação dos pacotes de tráfego relativos a reservas;**

- Na tabela A.6 encontram-se os testes efetuados para o requisito **Criação dos servidores fora das janelas temporais dedicadas para o envio de tráfego;**

- Na tabela A.7 encontram-se os testes efetuados para o requisito **Mecanismo de controlo de admissão de pedidos de reserva;**

- Na tabela A.8 encontram-se os testes efetuados para o requisito **Despacho de tráfego assíncrono na janela correspondente e segundo um critério de prioridades e capacidade dos servidores;**

Para sumariar os resultados obtidos durante o conjunto de testes que se encontram descritos em detalhe nas tabelas contidas na secção de anexos, encontra-se neste capítulo a tabela 6.2 que pretende assinalar a validação das funcionalidades criadas para cada requisito.

Requisito de teste	***Sumário dos Resultados Obtidos***
Validação do reencaminhamento de pacotes MSRP	Verificado o correto encaminhamento dos pacotes MSRP através dos módulos MRPBridge e MRPDaemon de cada porta.
Validação do mecanismo de filtragem e reserva de endereços MAC	Verificado o correto processamento dos pacotes e extração e armazenamento correto dos endereços MAC de origem a ser associados a cada porta.
Validação do mecanismo de localização de endereços MAC	Verificado o correto funcionamento do mecanismo implementado para localização da interface que contém armazenado endereços de MAC pretendidos.

Validação dos mecanismos de passagem de informação e comandos de controlo através do uso de memória partilhada	Verificado o correto funcionamento do mecanismo de troca de informações e comandos entre a MRPBridge e a camada *switching* do HaRTES.
Validação das estruturas implementadas para armazenamento do estado e características pertinentes às reservas MSRP segundo o FTT	Verificado o correto funcionamento e estabilidade da estrutura desenvolvida para gestão das reservas no sistema pela *thread mrp_control_event*.
Validação da criação de um servidor de background para cada interface do switch	Verificado o correto funcionamento dos servidores de *background* destinado a tráfego assíncrono não correspondente a qualquer reserva.
Validação da criação de servidores segundo as características das reservas	Verificado o correta criação de servidores segundo as especificações contidas nos pedidos de reserva MSRP.
Validação do mecanismo de diferenciação de tráfego- I	Verificado o correto funcionamento de alocação de pacotes nos servidores segundo a correspondência com as reservas criadas previamente no *switch*.
Validação do mecanismo de diferenciação de tráfego- II	Verificado o correto funcionamento de alocação de pacotes nos servidores segundo a correspondência com as reservas criadas previamente no *switch*, e no caso de não existir esta correspondência com qualquer reserva a alocação dos pacotes no servidor de *background* pertencente à interface que tem associada o endereço MAC de destino do pacote.

Validação das estruturas implementadas para armazenamento dos pedidos de reservas	Verificado o correto funcionamento e gestão das filas de pedidos de criação/destruição de servidores, criadas para cada interface.
Validação do mecanismo de criação de servidores fora das janelas temporais dedicadas a envio de tráfego	Verificado o processo de atendimento dos pedidos presentes nas filas de criação/destruição de servidores na *IDLE_WINDOW*.
Determinação do tempo de latência do switch	Verificado que no pior caso para um conjunto de 25 servidores diferentes o *switch* apresentou um tempo de latência máximo de 4497 ηs.
Validação do algoritmo que rege o controlo de admissão	Verificado que o algoritmo implementado é capaz de determinar corretamente os tempo de bloqueio e interferência consoante a origem e destino da *stream*, em concordância com os tempos calculados matematicamente na criação do conjunto de reservas existentes no *switch*.
Validação do mecanismo de gestão de capacidade dos servidores	Verificado que o módulo dos servidores planos é capaz de restringir o envio de tráfego de uma reserva específica segundo a capacidade declarada durante a sua criação e refrescamento dessa capacidade no período definido.

Tabela 6.2: Sumário dos resultados obtidos dos testes funcionais dos requisitos do sistema criado

6.4 Análise dinâmica do sistema

Para a verificação do comportamento dinâmico do sistema descrito no capítulo 5, foram utilizadas ferramentas de uma plataforma de análise dinâmica de aplicações designada de *Valgrind*, nomeadamente a aplicação *Callgrind* que analisa e regista informação acerca do conjunto de chamadas aos vários procedimentos de uma aplicação. A aplicação *Callgrind* é utilizada para criar um documento de texto que descreve características específicas. A aplicação *KCachegrind* é depois capaz de analisar este documento e fazer dois tipos de representações gráficas que serão utilizadas neste subcapítulo.

Para utilização destas ferramentas foi realizado o cenário de teste em que duas estações terminais atuando como *Talker* e *Listener* fazem uma reserva de recursos MSRP e procedem à comunicação de uma *stream* de vídeo transmitida através do *switch* HaRTES de duas portas. A ferramenta *Callgrind* foi aplicada apenas no processo referente à camada HaRTES. Do documento resultante através do *KCachegrind* foi possível gerar a figura 6.2, onde se encontra o *Callee Map* das principais *threads* do sistema. É possível verificar-se a ausência da *thread mrp_control_event*, que por comparação às restantes no sistema não é apresentada, isto é devido a ela só ser ativada nos instantes iniciais do cenário de teste, em que se procede aos pedidos de reserva de recursos MSRP. No entanto pode-se verificar com grande predominância a *thread* do *Multiplexer*, responsável pela distribuição dos pacotes provenientes de todas as interfaces do *switch* pelas filas e servidores correspondentes segundo os mecanismos criados. Dentro da *thread* do *Multiplexer* é possível também observar-se a chamada da função de introdução de pacotes no módulo dos servidores através da função *HorizontalServer_PutMsg* (a verde claro na figura 6.2) e a localização do servidor desejado através da função *ADV_MESSAGES_TABLE_MAC_ADD_SEARCH*. É possível também verificar na imagem as *threads Rx_event_handler* e *Tx_event_handler*, que são criadas para cada interface do *switch*, mas que nesta representação gráfica não é possível visualizar

esta criação por interface pois encontra-se indisponível a visualização da quantidade de *threads* com o mesmo nome. Podemos observar no *Rx_event_handler* a filtragem dos endereços MAC através da função *mac_queue_add*, e no *Tx_event_handler* a utilização da função *HorizontalServer_Dispatch.*

No entanto na outra representação disponível, *Call Graph,* apresentada na figura 6.3, tal informação relativa ao número *threads* já é possível de verificar, onde vemos a criação de três *threads* para eventos de envio e receção de pacotes. Duas destas *threads* serão então destinadas às interfaces desejadas do *Talker* e do *Listener*, e uma outra é da interface virtual para comunicação com os outros módulos do FTT que se encontram de momento desativados.

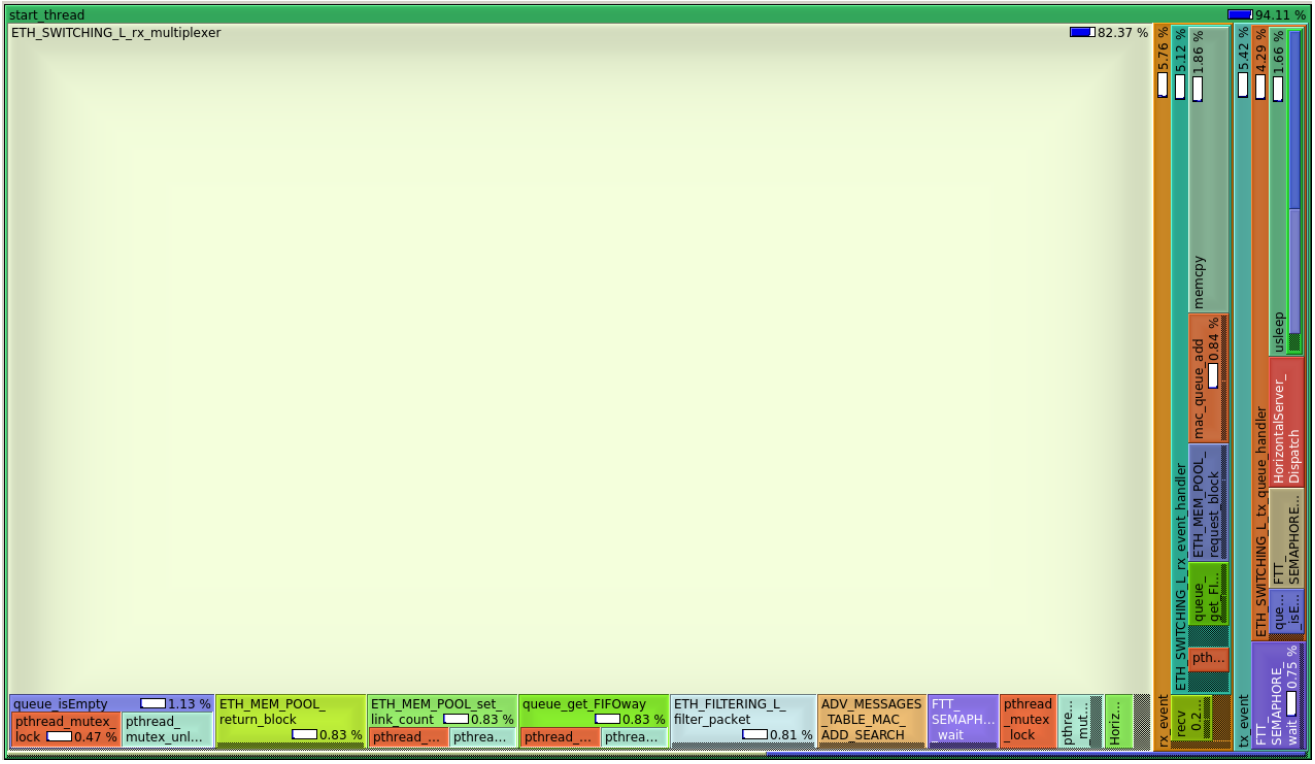

Figura 6.2: *Call Map* das *threads* principais do Sistema: *Multiplexer*, *Rx_event_handler* e *Tx_event_handler*

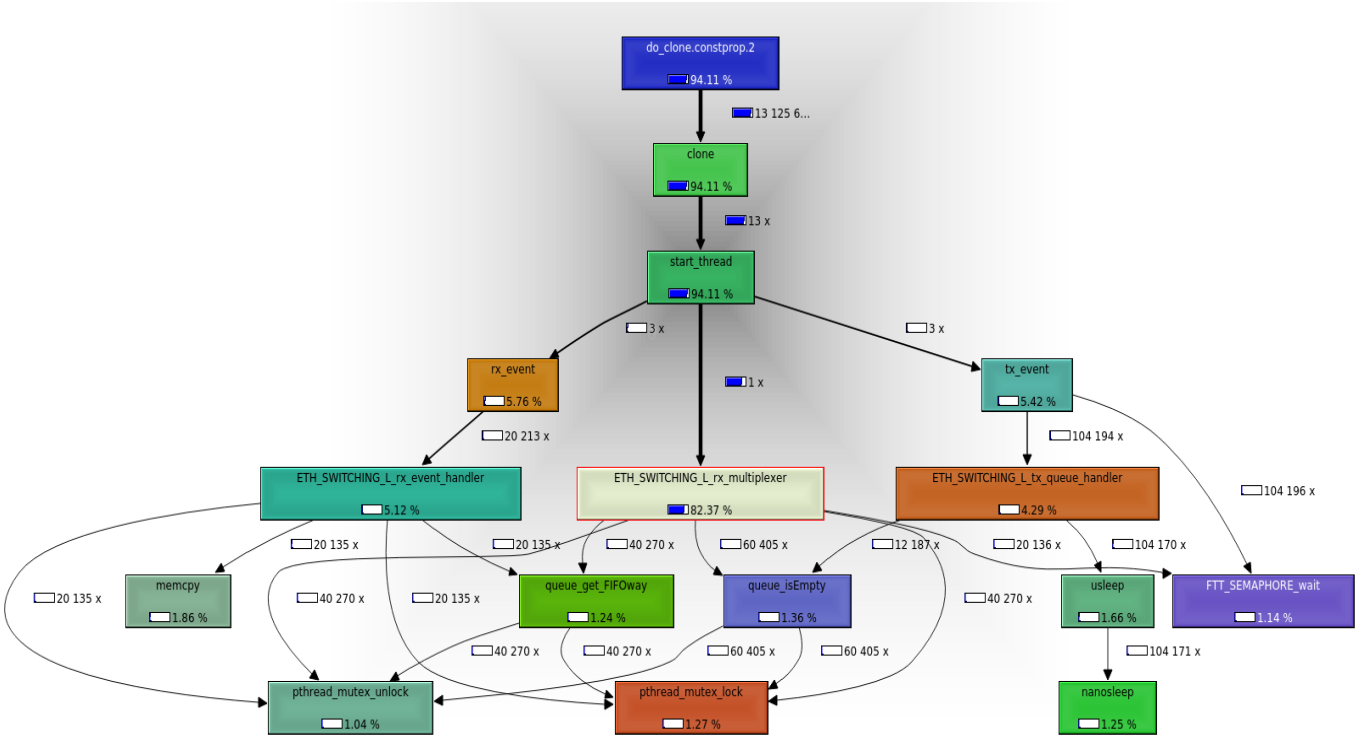

Figura 6.3: *Call Graph* das *threads* do *Multiplexer*, *Rx_event_handler* e *Tx_event_handler*

Em mais detalhe será também importante analisar os *Call Graphs* das *threads* correspondentes à máquina de estados que rege o EC, do envio e receção de pacotes, e por fim a *thread mrp_control_event* que efetua a gestão das tabelas de reservas no sistema e faz a comunicação com a MRPBridge.

Na figura 6.4 encontra-se o *Call Graph* representante do *Rx_event_handler*, as principais funções que convém notar é a filtragem dos endereços MAC à medida que estes chegam através da função *mac_queue_add*, e a obtenção de espaço nas filas de memória interna do *switch* através da função *ETH_MEM_POOL_request_block.* Existe também a interação com os *mutex*'s e semáforos do sistema de modo a proteger a informação nas filas de memória interna e fazer a sincronização com os restantes módulos do sistema.

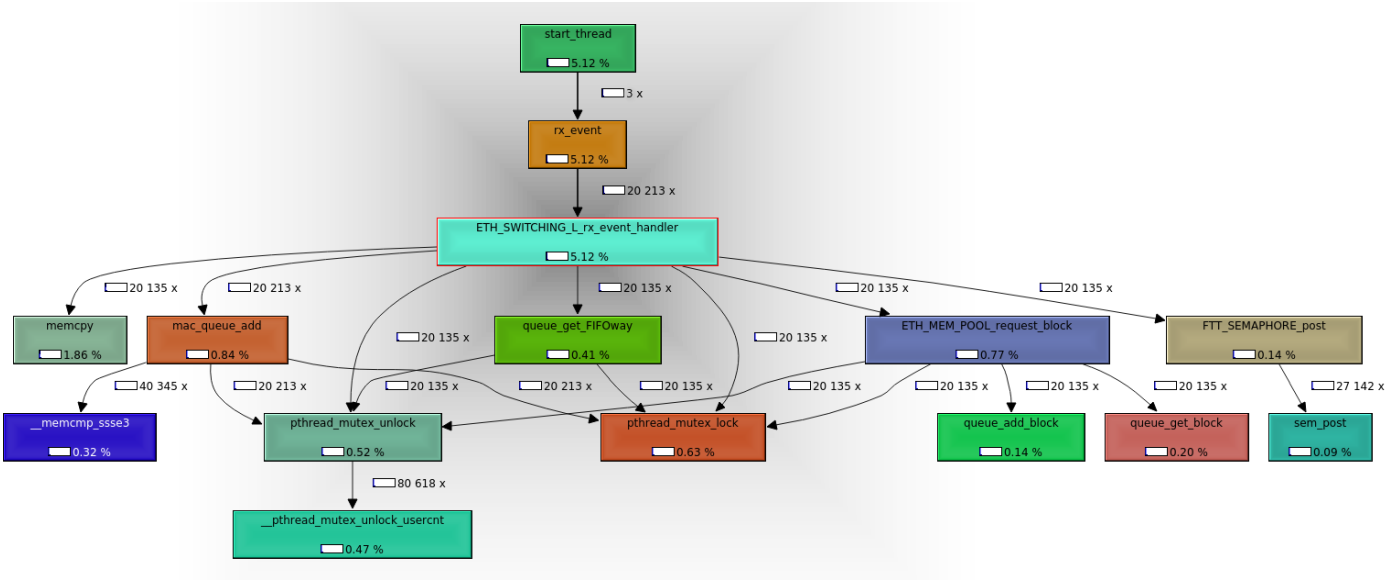

Figura 6.4: *Call Graph* do *Rx_event_handler*

Na figura 6.5 encontra-se o *Call Graph* representante da *EC_state_machine,* as principais funções que convém notar é o refrescamento da capacidade dos servidores dessa interface através da função *RefreshCapacity* e os mecanismos de sincronização com o relógio do sistema para definição das janelas temporais que definem o EC.

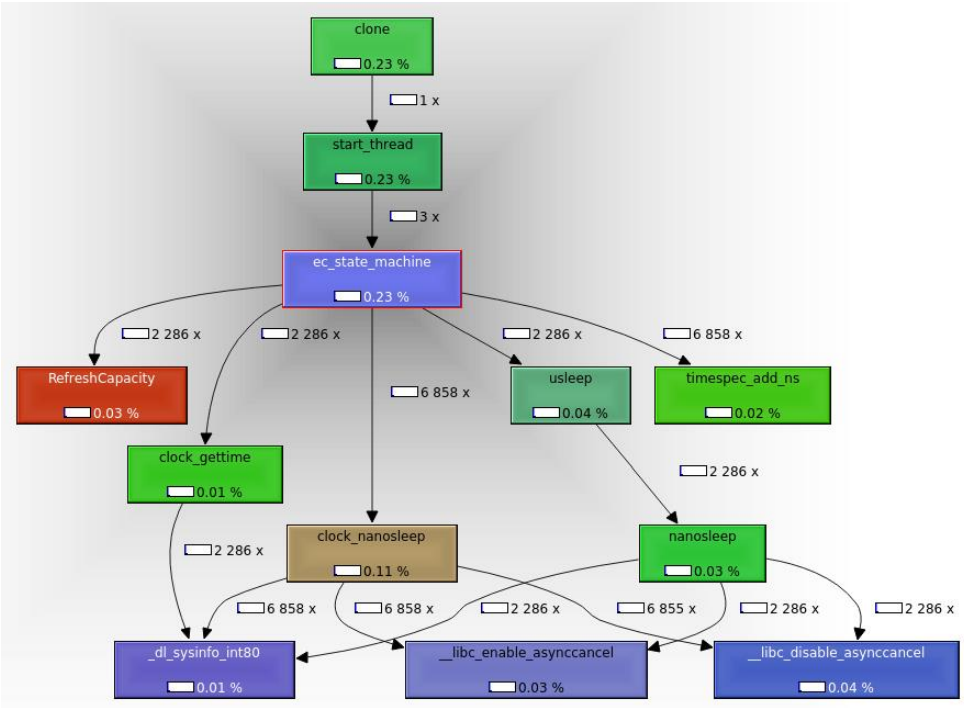

Figura 6.5: *Call Graph* da *EC_state_machine*

Na figura 6.6 encontra-se o *Call Graph* representante da *Tx_event_handler*, as principais funções que convém notar é o mecanismo de sincronização através de um semáforo que fica à espera que o *multiplexer* coloque um pacote nas suas filas ou nos servidores, e o despacho dos pacotes de tráfego assíncrono que são reservados no módulo dos servidores horizontais através da função *HorizontalServer_Dispatch*.

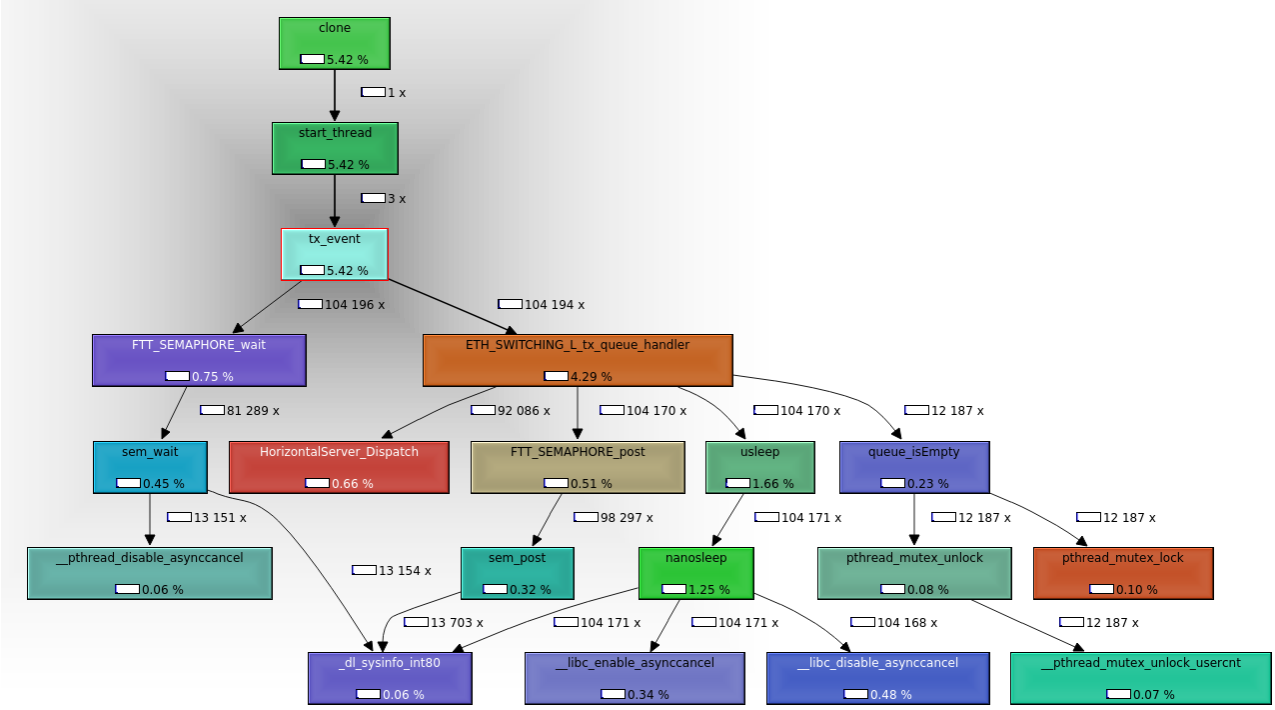

Figura 6.6: *Call Graph* do *Tx_event_handler*

Na figura 6.7 encontra-se o *Call Graph* representante da *mrp_control_event*, as principais funções que convém notar é obtenção da informação contida na memória partilhada, a gestão das tabelas que contêm a informação sobre o estado das reservas no sistema, os mecanismos de localização da interface correspondente ao endereço MAC de destino da *stream*, a gestão das filas de pedidos de criação/destruição de servidores, e por fim os mecanismos de sincronização e proteção da corrupção da informação no sistema.

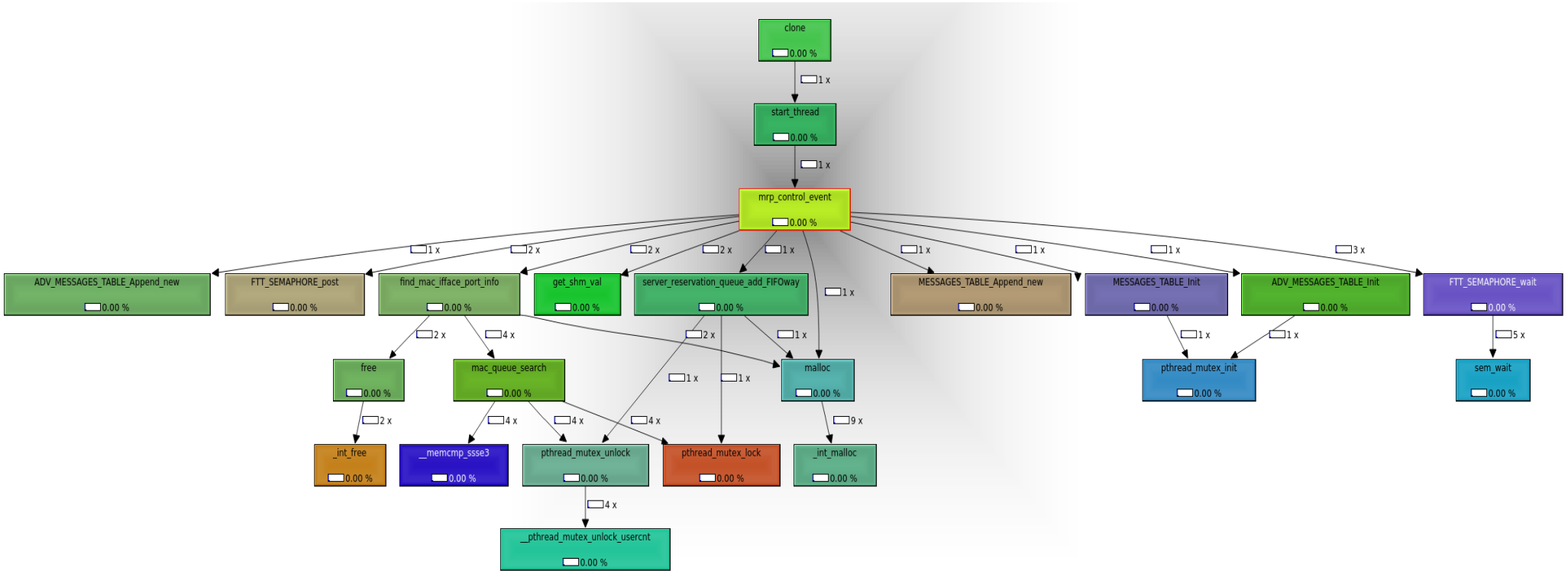

Figura 6.7: Call Graph do *Tx_event_handler*

Os resultados apresentam de uma forma geral uma predominância de funções que realizam operações a nível da memória do sistema, quer sejam estas efetuadas ao nível das filas de pacotes como ao nível da memória interna, proteção contra a corrupção de informação e sincronização de eventos, através das funcionalidades disponibilizadas pelos *mutex's* e semáforos. Isto justifica-se visto o sistema aqui representado ser um *switch* modificado de *Ethernet* onde a base do seu funcionamento assenta no reencaminhamento de pacotes entre as suas interfaces.

6.5 Análise do desempenho do sistema

Para verificar experimentalmente o funcionamento do sistema foram criados dois cenários de testes. O primeiro cenário prevê verificar experimentalmente os tempos do envio dos pacotes relativos a três reservas diferentes no sistema através de um histograma, e o segundo cenário prevê verificar o cumprimento da capacidade definida para três reservas através da análise da largura de banda usada por cada um dos servidores associados a estas reservas.

6.5.1 Teste com 3 servidores de períodos diferentes

No primeiro cenário de testes foi criado num *switch* de duas portas três reservas distintas com destino a uma estação terminal que funcionará como *Listener*. Todas estas reservas possuem a mesma capacidade de 4542 Bytes mas apresentam períodos diferentes. Recorrendo ao uso da ferramenta *PackEth* foram enviados através de uma outra estação terminal que funcionará como *Talker*, três tipos de pacotes diferentes. Os três tipos de pacotes diferem entre eles apenas no endereço MAC de origem, que foi associado cada um deles a uma reserva específica, e de tal modo a um servidor específico. A velocidade de envio de pacotes através do *PackEth* foi de 20 Mbit/s e todos os pacotes apresentam um tamanho fixo de 1514 Bytes e endereço MAC de destino o da estação terminal que

funcionará como *Listener.* Na estação terminal que funciona como *Listener* foi analisado, com recurso à ferramenta *Wireshark,* os pacotes de *Ethernet* que lhe chegam através da interface que conecta ao *switch.*

Na figura 6.7 encontra-se uma pequena captura da chegada dos pacotes *Ethernet*, está apresentado nesta figura os pacotes enviados durante a duração de dois ECs de 30ms onde durante o primeiro EC os três servidores apresentam capacidade para o envio de pacotes. Também se pretende demonstrar o despacho das mensagens segundo a prioridade dos servidores que têm as seguintes características: servidor com nível 9 de prioridade associado ao endereço MAC de origem começado por 0B com capacidade de 4542 Bytes e 1 EC de período, servidor com nível 6 de prioridade associado ao endereço MAC de origem começado por 0C com capacidade de 4542 Bytes e 2 ECs de período, e por fim servidor com nível 1 de prioridade associado ao endereço MAC de origem começado por 0A com capacidade de 4542 Bytes e 3 ECs de período.

No.	Time	Source	Destination	Protocol	Length	Info
2700	19.922853006	00:84:00:00:00:00	LLDP_Multicast	MRP-MSRP	35	Multiple Stream Reservation Protocol
2701	19.937745006	0b:00:00:00:00:00	LexmarkI_00:00:00	0x8ff0	1514	Ethernet II
2702	19.937758006	0b:00:00:00:00:00	LexmarkI_00:00:00	0x8ff0	1514	Ethernet II
2703	19.937769006	0b:00:00:00:00:00	LexmarkI_00:00:00	0x8ff0	1514	Ethernet II
2704	19.937778006	0c:00:00:00:00:00	LexmarkI_00:00:00	0x8ff0	1514	Ethernet II
2705	19.937788006	0c:00:00:00:00:00	LexmarkI_00:00:00	0x8ff0	1514	Ethernet II
2706	19.937799006	0c:00:00:00:00:00	LexmarkI_00:00:00	0x8ff0	1514	Ethernet II
2707	19.938015006	0a:00:00:00:00:00	LexmarkI_00:00:00	0x8ff0	1514	Ethernet II
2708	19.938642006	0a:00:00:00:00:00	LexmarkI_00:00:00	0x8ff0	1514	Ethernet II
2709	19.939167006	0a:00:00:00:00:00	LexmarkI_00:00:00	0x8ff0	1514	Ethernet II
2710	19.996482006	0b:00:00:00:00:00	LexmarkI_00:00:00	0x8ff0	1514	Ethernet II
2711	19.996496006	0b:00:00:00:00:00	LexmarkI_00:00:00	0x8ff0	1514	Ethernet II
2712	19.996510006	0b:00:00:00:00:00	LexmarkI_00:00:00	0x8ff0	1514	Ethernet II
2713	19.996624006	0a:00:00:00:00:00	LexmarkI_00:00:00	0x8ff0	1514	Ethernet II
2714	19.997148006	0a:00:00:00:00:00	LexmarkI 00:00:00	0x8ff0	1514	Ethernet II

Figura 6.7: Captura do *Wireshark* no *Listener*

Para analisar mais especificadamente os dados obtidos no *Wireshark* foi desenvolvida uma ferramenta no *MATLAB* para filtrar os pacotes recebidos segundo o primeiro campo do endereço MAC de origem e calcular a diferença de tempos de chegada entre pacotes referentes à mesma reserva. Com esta diferença pretende-se verificar que os servidores refrescam a sua capacidade nos instantes

certos e que quando existe capacidade disponível os servidores despacham os seus pacotes sequencialmente. Estas diferenças permitiram construir um histograma dos tempos de chegada dos pacotes segundo o seu servidor que permitem validar o comportamento apresentado na figura 6.8 que será o desejado quando os servidores estão dotados de controlo de capacidade e despacho segundo prioridades.

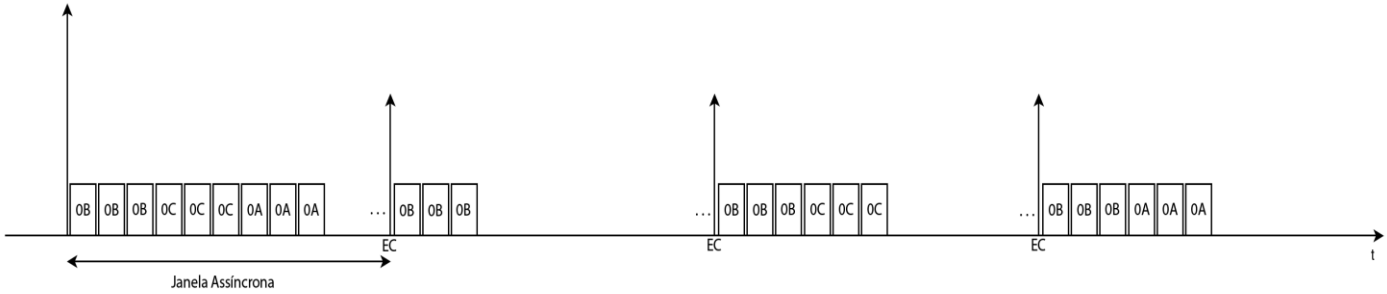

Figura 6.8: Comportamento desejado dos Servidores

Os histogramas das diferenças dos tempos de chegada dos pacotes recebidos correspondentes os três servidores encontra-se apresentado na figura 6.9:

- O primeiro corresponde ao servidor de maior prioridade cujo tráfego é proveniente do endereço MAC começado por 0B, é visível que o *switch* envia os pacotes sequencialmente enquanto o servidor possui capacidade, e quando esta não existe responde aos restantes servidores até que a capacidade deste seja refrescada.

- O segundo corresponde ao servidor de prioridade intermédia cujo tráfego é proveniente do endereço MAC começado por 0C, é visível o comportamento semelhante ao do histograma correspondente ao servidor de maior prioridade, no entanto a capacidade deste só é refrescada a cada dois ECs. A razão para o servidor não apresentar ser refrescado no instante exato de 60ms deve-se aos mecanismos de sincronização dos timers.

- O terceiro corresponde ao servidor de menor prioridade cujo tráfego é proveniente do endereço MAC começado por 0A, é visível o comportamento semelhante ao histograma do pacote de prioridade intermédia, mas no entanto devido a que de seis em seis ECs os pacotes deste servidor são despachados logo após os pacotes do servidor de maior prioridade, a diferença dos tempos acaba ser menor a 90ms entre os pacotes não enviados sequencialmente.

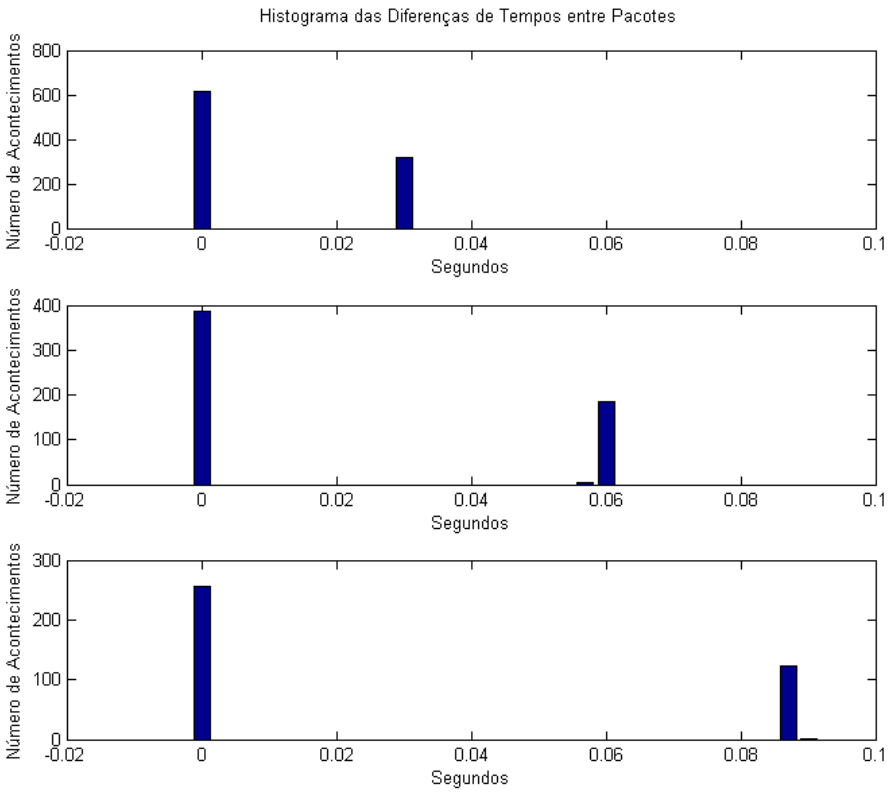

Figura 6.9: Histograma das diferenças dos tempos de chegada dos pacotes recebidos segundo servidores

6.5.2 Teste à largura de banda imposta pelos servidores

Para teste à largura de banda imposta pelos servidores foi concebido um cenário de teste onde foi criado no *switch* três reservas entre três *Talkers* distintos e um único *Listener*. Em semelhança ao teste do capítulo anterior instanciou-se o *switch* com duas portas e na estação terminal que funcionou como gerador de tráfego para os três *Talkers* foi utilizado a ferramenta *PackEth* para gerar pacotes

com endereços MAC de origem distintos, endereços MAC de destino correspondentes à estação terminal que assume o papel de *Listener*, e de tamanho fixo de 1514 Bytes. Na estação terminal que funcionou como *Listener* foi utilizado o *Wireshark* para captura dos pacotes recebidos.

Novamente foram utilizados os mesmos endereços MAC de origem que no teste anterior, estes foram associados a uma reserva de recursos no sistema e por consequência a servidores específicos para atenderem este tráfego. Foi desenvolvida no *MATLAB* uma ferramenta para analisar a captura efetuada pelo *Wireshark* e representar graficamente a largura de banda usada em cada EC.

A figura 6.10 representa a largura de banda usada por EC por cada um dos servidores:

- A linha azul corresponde ao servidor de maior prioridade cujo tráfego é proveniente do endereço MAC começado por 0A e que foi enviado a 40 Mbit/s, foi feita para esta *stream* uma reserva de recursos para 30 Mbit/EC. Como o tamanho máximo do pacote é de 1514 Bytes o sistema não prevê a divisão de pacotes e por isso criou um servidor com a capacidade de 113550 Bytes e com um período de um EC.

- A linha vermelha corresponde ao servidor de prioridade intermédia cujo tráfego é proveniente do endereço MAC começado por 0B e que foi enviado a 30Mbit/s, foi feita para esta *stream* uma reserva de recursos para 20 Mbit/EC. Como o tamanho máximo do pacote é de 1514 Bytes o sistema não prevê a divisão de pacotes e por isso criou um servidor com a capacidade de 75700 Bytes e com um período de um EC.

- A linha preta corresponde ao servidor de menor prioridade cujo tráfego é proveniente do endereço MAC começado por 0C e que foi enviado a 10 Mbit/s, foi feita para esta *stream* uma reserva de recursos para 10 Mbit/EC. Como o tamanho máximo do pacote é de 1514 Bytes o sistema não prevê a divisão de pacotes e por

isso criou um servidor com a capacidade de 37850 Bytes e com um período de um EC.

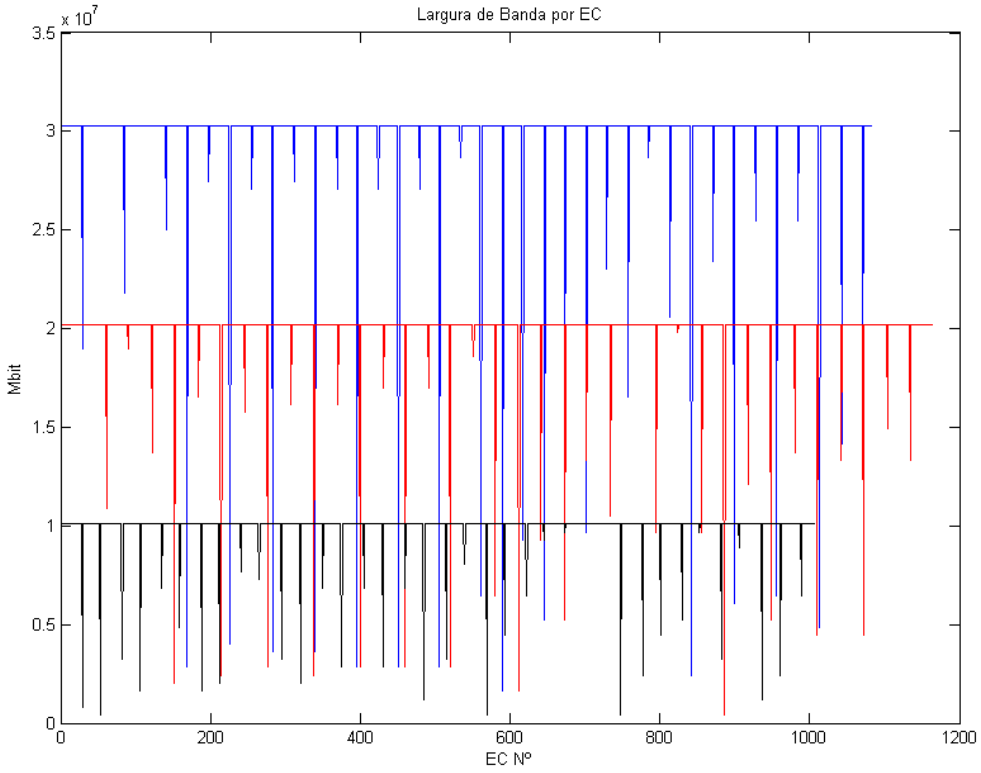

Figura 6.10: Análise da largura de banda usada por cada servidor

Para todas as reservas desejadas o sistema por defeito criou servidores com um pouco mais de capacidade para garantir sempre o envio de mais um pacote do tamanho mínimo declarado no pedido de reserva de recursos. Isto é visível no gráfico referente à largura de banda usada por cada servidor a cada EC. Os picos inferiores nos valores da largura de banda devem-se ao facto de que nas filas dos servidores durante aquele EC não se verificaram pacotes suficientes para utilizar toda a capacidade reservada. Verificou-se que os servidores e por consequência o *switch* é capaz de eficazmente fazer cumprir a largura de banda reservada para cada *stream* reservada através de uma gestão da capacidade e refrescamento desta por parte do módulo dos servidores horizontais e de uma diferenciação de tráfego segundo as reservas existentes no sistema e a estruturas montadas que asseguram esta funcionalidade de distinção de pacotes segundo reservas.

7. Conclusão

Apesar de ainda ser objeto de estudo o desenvolvimento e otimização da tecnologia *Ethernet* nas comunicações de Tempo-Real, o *switch* HaRTES apresenta-se como uma opção para quem deseja estabelecer uma rede, com a topologia em estrela, que forneça aos seus nós terminais garantias de Tempo-Real.

O *switch* HaRTES desenvolvido segundo o paradigma FTT permite a nós da rede compatíveis com o protocolo garantias de qualidade de serviço, no entanto a nós não compatíveis com o protocolo tal não é possível. Associado ao facto do *switch* HaRTES apenas possuir um mecanismo de reserva de recursos segundo o seu próprio protocolo, existiu a necessidade de criar um suporte para o protocolo comercial SRP. Querendo ainda manter a estrutura interna e os procedimentos particulares do HaRTES segundo o paradigma FTT, apenas seria necessário implementar o suporte para o mecanismo de sinalização de reservas de recursos segundo o SRP, descrito num dos protocolos que o compõem denominado por MSRP.

Seguindo o trabalho previamente realizado, que possibilitou a criação dos módulos de testes através da especificação dos fluxos a produzir pelas estações terminais segundo a estrutura das MSRPDUs, pretendeu-se com as atividades realizadas no âmbito do presente trabalho dotar o *switch* HaRTES com todos os mecanismos necessários para ser compatível com o MSRP.

Foram apresentados no presente documento uma descrição de todas as funcionalidades pertinentes, estruturas e processos implementados que permitem o *switch* HaRTES realizar tanto o encaminhamento correto das mensagens protocolares do MSRP como também das mensagens relativas às *streams* para as quais foram realizadas reservas de recursos, bem como reservar de recursos graças à criação de servidores, realizar um controlo de admissão de tráfego e reservas e o escalonamento dos pacotes das *streams* de tráfego aperiódico segundo as mecânicas dos *Deferrable Servers*.

O trabalho realizado foi concebido tendo em conta certos requisitos necessários para o correto funcionamento do *switch* HaRTES. Estes encontram-se descritos no capítulo 6 juntamente com um sumário dos resultados obtidos nos testes realizados com vista à validação dos requisitos impostos. A maioria dos requisitos foram cumpridos com sucesso, no entanto, ainda se verifica a necessidade de adaptar todo o sistema para ser capaz de ser implementado numa rede com uma arquitetura *Multi-Hop.* Desta forma o sistema criado apenas se encontra validado para o caso em que a topologia da rede se encontra em estrela pois não possui o mecanismo de sincronização segundo o protocolo IEEE 802.1AS que permitiria aos elementos de rede no cenário *Multi-Hop* calcular o atraso ao longo da rede.

Por fim o sistema foi testado externamente, com recurso a diversas ferramentas para obtenção de valores e tratamento de dados, de forma a provar que o sistema cumpre os requisitos inicialmente definidos. Verificou-se que o sistema é capaz de realizar o reencaminhamento de pacotes segundo o seu destinatário, de realizar a diferenciação de tráfego segundo as reservas existentes no sistema, que é capaz de autonomamente realizar e desfazer reservas, fazendo sempre uma gestão do tráfego que é despachado pelo módulo dos servidores planos sem que nunca exceda a capacidade criada no instante da reserva de recursos.

7.1 Trabalho Futuro

O trabalho realizado apesar de dotar o *switch* HaRTES das capacidades fundamentais para suportar o protocolo MSRP, não inclui suporte as restantes normas. Para total suporte do protocolo será necessário o *switch* HaRTES também possuir adicionalmente mecanismos que garantam o seu correto funcionamento segundo as restantes normas, sendo estas o MVRP e o MMRP que possibilitam a criação de VLAN's dentro da rede e a propagação de pedidos de reserva segundo endereços MAC.

É também necessário mencionar que o trabalho desenvolvido não teve em conta uma situação em que a rede pode conter mais que um *switch* HaRTES e assim tratar-se de uma rede *Multi-Hop*. Desta forma como trabalho futuro sugere-se que se adapte o sistema de forma a lidar com tal situação. As áreas onde é possível facilmente prever a necessidade de posterior desenvolvimento são:

- Algorítmo usado no controlo de admissão de pedidos de reserva, onde o rastreio das reservas pertinentes para o cálculo dos tempos de resposta no *Uplink* e *Downlink* é realizado apenas por endereços MAC e não por interface de entrada e de saída, visto a rede de desenvolvimento ter sido imaginada segundo uma topologia em estrela;

- Mecanismo de cálculo de latência ao longo de uma rede *Multi-Hop*, onde é importante para garantir que as reservas efetuadas tomam em conta eventuais atrasos que se podem propagar ao longo da rede;

- Aprofundamento e otimização do mecanismo de passagem de valores segundo o protocolo MSRP para os valores característicos do FTT possibilitando uma melhor definição das reservas de recursos a realizar;

- Arquitetura hierárquica de servidores, uma vez que foi inicialmente pensada mas não concebida, e que possibilitaria um controlo mais otimizado do tráfego segundo certas características;

8. Referências

[1] G. C. Buttazzo. *Hard Real-time Computing Systems: Predictable Scheduling Algorithms And Applications*. Springer, 2011.

[2] P. Pedreiras, "Temporal constraints: source and characterization, Apontamentos da Disciplina de Sistemas de Tempo-Real", ed: Universidade de Aveiro, 2014.

[3] P. Pedreiras, "Introduction to Real-Time kernels, Apontamentos da Disciplina de Sistemas de Tempo-Real", ed: Universidade de Aveiro, 2014.

[4] P. Pedreiras, "Computational Models, Apontamentos da Disciplina de Sistemas de Tempo-Real", ed: Universidade de Aveiro, 2014.

[5] P. Pedreiras, "Aperiodic task scheduling, Apontamentos da Disciplina de Sistemas de Tempo-Real", ed: Universidade de Aveiro, 2014.

[6] J. P. Lehoczky, L. Sha, and J. K. Strosnider. "Enhanced aperiodic responsiveness in hard real-time environments". *Proceedings of the IEEE Real-Time Systems Symposium,* December 1987.

[7] B. Sprunt, L. Sha, and J. Lehoczky. "Aperiodic task scheduling for hard-real-time systems". *Journal of Real-Time Systems,* 1, July 1989.

[8] J.K Strosnider, J.P. Lehoczky, and L. Sha. "The deferrable server algorithm for enhancing aperiodic responsiveness in hard-real-time environments". *IEEE Transactions on Computers,* 4(1), January 1995.

[9] J. D. Decotignie, "Ethernet-based real-time and industrial communications," Proceedings of the IEEE, vol. 93, pp. 1102-1117, 2005.

[10] *General Purpose Field Communication System, vol. 3/3 (WorldFIP),* CENELEC Std. EN 50170, 1996.

[11] *General Purpose Field Communication System, vol. 2/3 (Profibus),* CENELEC Std. EN 50170, 1996.

[12] *General Purpose Field Communication System, vol. 1/3 (P-NET),* CENELEC Std. EN 50170, 1996.

[13] *High Efficiency Communication Subsystem for Small Data Packages,* CENELEC Std. EN 50254, 1998.

[14] *Low-Voltage Switchgear and Controlgear—Controller-Device Interfaces (CDI's)–Part 2: Actuator Sensor Interface* (AS-i), IEC Std. 62 026-2, 2000.

[15] *Electrical Equipment of Industrial Machines—Serial Data Link for Real-Time Communication Between Controls and Drives*, IEC Std. 61 491, 2002.

[16] *Control Network Specification*, Electron. Ind. Alliance Std. EIA-709.1, Mar. 1998.

[17] H. Kirrmann and P. Zuber, "The IEC/EEE train communication network," IEEE Micro, vol. 21, no. 2, pp. 81–92, Mar.–Apr. 2001.

[18] M. Haverty, "MIL-STD 1553—A standard for data communications," *Commun. Broadcasting*, vol. 10, no. 1, pp. 29–33, Jan. 1986.

[19] *Low-Voltage Switchgear and Controlgear—Controller-Device Interfaces (CDI's)—Part 5: Smart Distributed System (SDS)*, IEC Std.62 026-5, 2000.

[20] *Low-Voltage Switchgear and Controlgear—Controller-Device Interfaces (CDI's)—Part 3: DeviceNet*, IEC Std. 62 026-3, 2000.

[21] *Road Vehicles—Exchange of Digital Information—Controller Area Network (CAN) for High-Speed Communication*, ISO Std. 11 898, 1993.

[22] IEEE Standards Association. *IEEE 802.3™ Ethernet*, Disponível: https://standards.ieee.org/about/get/802/802.3.html, [12 Setembro 2015]

[23] CISCO DocWiki. *Ethernet Technologies*, Disponível: http://docwiki.cisco.com/wiki/Ethernet_Technologies, [12 Setembro 2015]

[24] *Carrier Sense Multiple Access Collision Detect (CSMA/CD)* Explained.

[25] R. G. V. Santos, "Enhanced ethernet switching technology for adaptive hard real-time applications" Departamento de Eletrónica, Telecomunicações e Informática, Universidade de Aveiro, 2011.

[26] P. Pedreiras, R. Leite, and L. Almeida, "Characterizing the real-time behavior of prioritized switched-ethernet," *2nd RTLIA*, 2003.

[27] R. Marau, L. Almeida, and P. Pedreiras, "Enhancing real-time communication over COTS Ethernet switches," 2006, pp. 295-302.

[28] R. Marau, P. Pedreiras, and L. Almeida. "Asynchronous Traffic Signaling over Master-Slave Switched Ethernet Protocols." *Proceedings of the 6th International Workshop on Real-Time Networks (RTN'07)*, July 2007.

[29] L. M. T. Nóbrega, "Suporte MSRP para Hard QdS Switch", Departamento de Eletrónica, Telecomunicações e Informática, Universidade de Aveiro, 2012.

[30] "IEEE Standard for Local and Metropolitan Area Networks – Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks," IEEE Std 802.1Q™-2011 (Revision of IEEE Std 802.1Q-2005), 2011.

[31] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, "RSVP: a new resource reservation protocol", *IEEE Communications Magazine*, vol. 40, pp, 116-127, 2002.

[32] R. Sousa, P. Pedreiras and P. Gonçalves, "Enabling IIoT IP backbones with real-time guarantees", *IEEE International Conf. On Emerging Technologies and Factory Automation – ETFA*, Sep. 2015

[33] FTT Flexible Time-Triggered Communications, Disponível: http://paginas.fe.up.pt/~ftt/sections/Flavours/index.html, [18 Dezembro 2014]

[34] M. Ashjaei, L. Silva, M. Behnam, P. Pedreiras, R. J. Brill, L. Almeida and T. Nolte, "Improved Message Forwarding for Multi-Hop HaRTES Real-Time Ethernet Networks", *Journal of Signal Processing Systems* 81(1939-8115), 1-21, 2015

9. Anexos

Identificação do requisito de teste	*Descrição do cenário de teste*	*Descrição dos resultados obtidos*
Validação do reencaminhamento de pacotes MSRP	Foram inicializados os processos de declaração de pedidos de reserva MSRP por parte de duas estações terminais atuando como *Talkers* enquanto uma terceira estação funcionou como *Listener*. Os *Talkers* declararam os pedidos de reserva e o *switch* HaRTES aceitou incondicionalmente os pedidos e propagou-os pela rede com o intuito de achar os seus *Listeners*. O *Listener* respondeu positivamente aos pedidos de reserva MSRP esta resposta será propagada em direção aos *Talkers*. Após o estabelecimento da reserva MSRP os *Talkers* desfazem a reserva emitindo novas declarações. Este processo foi repetido 100 vezes por ambos os *Talkers*. Para observação do sistema foi utilizado o *Wireshark* e as consolas dos processos relativos.	A filtragem dos pacotes MSRP foi verificada através do uso da ferramenta *Wireshark* onde foi possível verificar o encaminhamento destes pacotes através dos módulos MRPBridge e MRPDaemon de cada porta obedecendo ao procedimento indicado no protocolo MSRP. Adicionalmente foram analisadas as filas de memória da camada de *Ethernet switching* do HaRTES que após a receção de um pacote com o identificador MSRP que se encontraram vazias.
	Notas: N.A.	

Tabela A.1: Testes realizados para a validação do requisito Filtragem e gestão de pacotes MSRP

Identificação do requisito de teste	*Descrição do cenário de teste*	*Descrição dos resultados obtidos*
Validação do mecanismo de filtragem e reserva de endereços MAC	Nas estações terminais foram criadas através da ferramenta PackEth, pacotes com o campo de endereço MAC de origem com sequências específicas de forma a analisar o funcionamento da MAC *Address Fowarding Table*. As sequências estarão identificadas com um identificador da estação no próprio endereço MAC e estes irão repetir-se ao longo da sequência. Será analisado se à chegada de um pacote numa determinada porta, a *thread* responsável pela gestão do evento é capaz de filtrar o endereço MAC desejado, e armazenar este para posteriores ações.	A filtragem dos endereços MAC de origem foi analisada através da impressão da lista de endereços contidas na tabela de cada uma das portas dos *switch.* Verificou-se também que não existiam entradas repetidas nas tabelas, e que cada tabela apenas apresentavam os endereços MAC de origem correspondentes às estações terminais acessíveis através da sua porta correspondente.
	Notas: Através deste teste foi possível também validar em específico a funcionalidade de inicialização autónoma das MAC *Address Fowarding Table* para cada uma das portas inicializadas no *switch*, a funcionalidade de adição de novos endereços MAC às tabelas, a gestão das tabelas para que não sejam preenchidas com múltiplos endereços MAC repetidos, e por fim a funcionalidade que permite a qualquer altura visualizar o conteúdo destas tabelas.	

Validação do mecanismo de localização de endereços MAC	Foi criado um cenário de teste onde em cada uma das tabelas de endereços MAC existem 10 endereços únicos. Através da interação disponibilizada através das consolas, após o preenchimento de 30 endereços MAC únicos no total das três portas inicializou-se a funcionalidade de localização de endereços MAC específicos, sendo utilizados como referências três endereços que estarão presentes em cada uma das tabelas e um outro endereço que não estará presente em nenhuma delas.	Verificou-se que no caso de os endereços estarem presentes nas tabelas, estes são localizados e é possível através deste mecanismo aceder agora a toda a informação pertinente à porta pela qual este endereço é acessível. No caso de o endereço não estar presente o mecanismo exibe o comportamento desejado notificando de tal ocorrência.
	Notas: Através deste teste é possível validar para posteriores ações o mecanismo de obtenção de acesso às estruturas e informações de cada interface segundo a passagem de um endereço de MAC específico.	

Tabela A.2: Testes realizados para a validação do requisito Filtragem e gestão de endereços MAC

Identificação do requisito de teste	*Descrição do cenário de teste*	*Descrição dos resultados obtidos*
Validação dos mecanismos de passagem de informação e comandos de controlo através do uso de memória partilhada	Foi concebido um teste em que duas estações terminais, uma atuando como *Talker* e outra atuando como *Listener* realizam sucessivos ciclos de um pedido de reserva e a sua destruição de recursos, e onde *switch* HaRTES aceita incondicionalmente todos estes pedidos. O momento que corresponde à funcionalidade do MAP no MSRP ocorre no processo MRPBridge, no entanto devido ao design do sistema será necessário passar as características do pedido de reserva para o processo correspondente ao *switch* HaRTES através de uma estrutura em memória partilhada, para que este verifique o estado das reservas já existentes e se é possível garantir qualidade de serviço para esta nova reserva. O processo do *switch* HaRTES irá indicar sempre à MRPBridge respostas com um carácter positivo para a realização e destruição de reservas. Este processo foi repetido 100 vezes e para observação do sistema foram utilizadas as consolas dos processos relativos.	Verificou-se a correta passagem das características intervenientes nos critérios de decisão característicos do MAP, extraídas através das mensagens protocolares do MSRP. Verificou-se a correta extração dos valores característicos do campo *FirstValue* por parte da MRPBridge, o preenchimento dos campos criados para estes em memória partilhada e a passagem de comandos relativa às ações a realizar, seja esta de pré-reserva, reserva ou destruição desta. Verificou-se também a leitura correta dos campos preenchidos na memória partilhada por parte do processo do *switch* HaRTES e a correta passagem de comandos afirmativa através da memória partilhada.
	Notas: Através deste teste foi possível também validar o correto acesso a este recurso partilhado, cuja sincronização foi implementada com o uso de dois semáforos para o acesso à memória partilhada.	

Validação das estruturas implementadas para armazenamento do estado e características pertinentes às reservas MSRP segundo o FTT	Foi concebido um teste em que duas estações terminais atuando como *Talkers* e outra estação terminal atuando como *Listener* realizam reservas de recursos no *switch* HaRTES. Após cada um dos *Talkers* realizarem 25 reservas de recursos com *Unique IDs* e outros campos não responsáveis pela origem e destino da stream distintos, para a qual se reservam recursos, estes procedem a desfazer as reservas criadas. O *switch* HaRTES irá aceitar incondicionalmente todos os comandos emitidos pela MRPBridge e irá realizar as operações desejadas sobre as estruturas de acordo com estes. Para observação do sistema foram utilizadas as consolas dos processos MRPBridge e *switch* HaRTES	Verificou-se o correto funcionamento e estabilidade da estrutura *adv_SRDB* concebida para a camada *Ethernet switching* do HaRTES e a correta gestão desta estrutura por parte da *thread mrp_control_event* assim como da estrutura original das reservas de recursos do *switch*. Apurou-se o correto funcionamento através da verificação do conteúdo da tabela que armazena as características das reservas, o que revelou a correspondência dos campos presentes na estrutura *ADV_MESSAGE_REQUESITS* com os valores atribuídos às características das reservas MSRP. Isto verificou-se após os *Talkers* realizarem todas as reservas e também após estes as desfazerem as reservas, o que resultou numa tabela sem nenhum item.
	Notas: Através deste teste foi possível também validar em específico o funcionamento geral da *thread* que gere os eventos que originam da propagação das mensagens protocolares MSRP e da passagem destes por parte da MRPBridge para posterior análise e processamento. Também valida em específico a funcionalidade de adição de novas entradas na tabela destinada ao armazenamento do estado e características das reservas no sistema, a remoção de um item desta tabela através da localização do *StreamID* correspondente e identificador da sua reserva, e os mecanismos de proteção da corrupção de dados implementados com *mutex's*, e por fim a funcionalidade que permite a qualquer altura visualizar o conteúdo desta tabela.	

Tabela A.3: Testes realizados para a validação do requisito Compatibilidade e modularidade das estruturas e mecanismos FTT com suporte para MSRP

Identificação do requisito de teste	*Descrição do cenário de teste*	*Descrição dos resultados obtidos*
Validação da criação de um servidor de background para cada interface do switch	Foi concebido um teste em que uma estação terminal com o uso de uma aplicação de vídeo gera um fluxo na entrada de uma porta do *switch*. Nas restantes estações terminais realizou-se a receção do *stream* de vídeo. Para observação interna do sistema expôs-se o número de *slots* usados e disponíveis da *Ethernet memory pool* do *switch* no início de cada EC assim como o número de pacotes presentes no *background server* de cada porta.	Verificou a estabilidade do sistema através da qualidade de receção do vídeo, e o correto funcionamento dos servidores através do preenchimento das *Ethernet memory pool* do *switch* e de cada *background server* que apresentavam um valor de metade de pacotes em relação aos *slots* de pacotes usados na *memory pool*.
	Notas: Através deste teste foi possível validar a criação interna de uma *ServerTable,* que contem um número de entradas de *ServerTableItem's* correspondente ao número de interfaces do *switch,* e a criação autónoma do servidor de maior prioridade *HP_server* que no arranque do sistema são os *background servers* para cada uma das portas. Também foi possível validar a funcionalidade de alocação de pacotes nestes servidores à medida que estes são processados pela *thread rx_multiplexer* através do uso dos identificadores correspondentes à porta de saída e do servidor desejado dessa porta. Adicionalmente também validou a funcionalidade de *dispatching* de pacotes contido num servidor.	

Validação da criação de servidores segundo as características das reservas	Foi concebido um teste em que duas estações terminais atuando como *Talkers* e outra estação terminal atuando como *Listener* realizam reservas de recursos no *switch* HaRTES. Após cada um dos *Talkers* realizarem 25 reservas de recursos com *Unique IDs* distintos, para a qual o sistema irá criar servidores na porta correspondente ao *Listener*, estes procedem após todas os servidores criados a desfazer as reservas criadas e por sua vez também os servidores. O *switch* HaRTES irá aceitar incondicionalmente todos os comandos emitidos pela MRPBridge e irá realizar as operações desejadas sobre as estruturas de acordo com estes. Para observação do sistema foi utilizada a consola do processo correspondente ao *switch* HaRTES para imprimir as características dos servidores existentes na *ServerTable.*	Verificou-se que após cada atendimento a um pedido de reserva por parte da *thread mrp_control_event* tinha sido criado um novo servidor no *ServerTableItem* da porta de destino do *Listener*. Após todas as reservas serem criadas foi possível verificar-se que os 51 servidores se encontravam ligados na sequência desejada segundo as suas prioridades e instante de criação, finalizando com o *background server* característico de cada interface. Também se verificaram as corretas correspondências das características dos servidores com as dos pedidos de reserva MSRP. Em semelhança também se verificou a remoção dos servidores à medida que os *Talkers* iam desfazendo as reservas.
	Notas: Através deste teste foi possível validar a criação de servidores com o intuito de armazenarem os pacotes relativos às *streams* das reservas MSRP e a sua gestão de forma a encontrarem-se ordenados segundo as suas prioridades, apresentando como acesso à lista ligada de servidores sempre o servidor mais prioritário e finalizando a lista no *background server* da respetiva porta. Adicionalmente também foi possível validar a funcionalidade de remoção de servidores segundo um *ServerID* e um identificador do *ServerTableItem*.	

Tabela A.4: Testes realizados para a validação do requisito Capacidade de armazenamento de pacotes em servidores dedicados a reservas específicas

Identificação do requisito de teste	***Descrição do cenário de teste***	***Descrição dos resultados obtidos***
Validação do mecanismo de diferenciação de tráfego - I	Foi concebido um cenário de teste onde duas estações terminais funcionando como *Talkers* realizam cada uma delas uma reserva de recursos para comunicação de uma *stream* de dados com destino a uma terceira estação que irá funcionar como *Listener*. Após a confirmação da reserva os *Talkers* através da ferramenta *PackEth* vão gerar pacotes com o endereço MAC de destino do *Listener* e o de origem do *Talker* correspondente. Serão gerados 50 pacotes por cada um dos *Talkers* à velocidade máxima disponibilizada pela ferramenta. Encontra-se desativado o despacho de pacotes no *switch.* Será analisado com recurso à consola correspondente do *switch* HaRTES a distribuição do tráfego segundo os servidores correspondentes às reservas realizadas.	Verificou-se que após a confirmação do pedido de reserva por parte dos *Talkers* o sistema agora possuía dois servidores criados especificamente para estas reservas na interface do *switch* correspondente à do *Listener* para além do *background server* característico de cada porta. Após o envio de todos os pacotes por parte dos *Talkers* averiguou-se o conteúdo dos três servidores. Verificou-se que o *background server* se encontrava vazio, e que os servidores criados para as reservas apenas continham os 50 pacotes criados para o respetivo tráfego entre as duas estações terminais. Também convém mencionar que o tráfego contido nestes servidores se apresentam ordenados segundo FCFS.
	Notas: Através deste teste foi possível testar a capacidade de diferenciação de tráfego segundo os campos de endereços MAC contidos nos pacotes de *Ethernet* e os contidos na tabela *adv_SRDB* que armazena toda a informação pertinente relativa às reservas no sistema. Também foi possível validar os mecanismos de procura e identificação de reservas e posterior armazenamento de tráfego.	

Validação do mecanismo de diferenciação de tráfego - II	Foi concebido um cenário de teste onde duas estações terminais funcionando como *Talkers* em que apenas uma dela realiza a reserva de recursos para comunicação de uma *stream* de dados com destino a uma terceira estação que irá funcionar como *Listener*. Após a confirmação da reserva os *Talkers* através da ferramenta *PackEth* vão gerar pacotes com o endereço MAC de destino do *Listener* e o de origem do *Talker* correspondente. Serão gerados 50 pacotes por cada um dos *Talkers* à velocidade máxima disponibilizada pela ferramenta. Encontra-se desativado o despacho de pacotes no *switch.* Será analisado com recurso à consola correspondente do *switch* HaRTES a distribuição do tráfego segundo os servidores correspondentes às reservas realizadas.	Verificou-se que após a confirmação do pedido de reserva apenas por parte de um *Talker* o sistema agora possuía um servidor criado especificamente para esta reserva na interface do *switch* correspondente à do *Listener* para além do *background server* característico de cada porta. Após o envio de todos os pacotes por parte dos *Talkers* averiguou-se o conteúdo dos dois servidores. Verificou-se que o *background server* se encontrava com o tráfego proveniente da estação cuja reserva de recursos não havia sido realizada, e que o servidor criado para a reserva apenas continha os 50 pacotes criados para o respetivo tráfego entre as duas estações terminais. Também convém mencionar que o tráfego contido em ambos os tipos de servidores se apresentam ordenados segundo FCFS.
	Notas: Através deste teste foi possível testar a capacidade de diferenciação de tráfego segundo os campos de endereços MAC contidos nos pacotes de *Ethernet* e os contidos na tabela *adv_SRDB* que armazena toda a informação pertinente relativa às reservas no sistema. Também foi possível validar os mecanismos de procura e identificação de reservas e posterior armazenamento de tráfego segundo os resultados obtidos, onde se verificou que o tráfego proveniente de uma estação terminal que não realizou reservas de recursos é armazenado no *background server.*	

Tabela A.5: Testes realizados para a validação do requisito Diferenciação dos pacotes de tráfego relativos a reservas

Identificação do requisito de teste	*Descrição do cenário de teste*	*Descrição dos resultados obtidos*
Validação das estruturas implementadas para armazenamento dos pedidos de reservas	Foi concebido um cenário de teste onde uma estação terminal funciona como *Talker* e outras duas como *Listeners.* O *Talker* realiza 5 pedidos de reservas de recursos distintas para cada um dos *Listeners* e após a aceitação do pedido a posterior destruição da reserva. Para observação do sistema foi utilizada a consola do processo correspondente ao *switch* HaRTES para imprimir as características dos pedidos de reserva existentes em cada *server_reservation_queue* de cada interface do *switch* correspondente às estações terminais funcionando como *Listeners.*	Verificou-se o correto funcionamento da estrutura *server_reservation_queue* concebida para a criação e destruição de servidores na camada *Ethernet switching* do HaRTES fora das janelas temporais para envio de tráfego e a correta gestão desta estrutura por parte da *thread mrp_control_event*. Apurou-se o correto funcionamento através da verificação do conteúdo da das filas *server_reservation_queue*, o que revelou a correspondência dos campos presentes na estrutura *reservation_data* com as *ADV_MESSAGE_REQUESITS* da *adv_SRDB* e os valores atribuídos às características das reservas MSRP. Isto verificou-se após o *Talker* realizar todos os pedidos de reservas e também após este desfazer estas. Resultando em duas filas nas interfaces dos *Listeners* com dados para realizar comandos no módulo dos servidores planos segundo os respetivos endereços MAC de destino.
	Notas: Através deste teste foi possível validar as funcionalidades de criação de *reservation_data*, os mecanismos de proteção da corrupção de dados implementados com *mutex's* nas filas, a respetiva adição de entradas nas filas *server_reservation_queue* segundo o endereço MAC de destino da reserva, e por fim a funcionalidade que permite a qualquer altura visualizar o conteúdo desta tabela.	

Validação do mecanismo de criação de servidores fora das janelas temporais dedicadas a envio tráfego	Foi concebido um cenário de teste onde duas estações terminais funcionam como *Talker* e uma outra como *Listener*. Cada *Talker* realiza 50 pedidos de reservas de recursos distintos para o *Listener* e após a aceitação do pedido a posterior destruição da reserva. Para observação do sistema foi utilizada a consola do processo correspondente ao *switch* HaRTES para guardar o conteúdo da *server_reservation_queue* de cada interface do *switch* correspondente às estações terminais funcionando como *Listeners* e o estado dos servidores de cada uma destas interfaces.	Verificou-se o correto funcionamento do mecanismo de criação fora das janelas temporais indesejadas. Isto foi possível através da análise do comportamento das *threads* que gerem os eventos de envio de pacotes (*tx_queue_handler*) de cada porta. Estas *threads* com uma máquina de estados regida segundo a *EC_state_machine* apenas realizaram a criação e destruição de servidores durante o estado correspondente ao *IDLE_WINDOW*. No entanto verificou-se que na gerência deste mecanismo, tanto a leitura dos comandos armazenados nas filas *server_reservation_queue* ou gestão do módulo dos servidores planos pode ultrapassar, pode ultrapassar a *IDLE_WINDOW* e passar para a *TM_WINDOW*. Durante o teste foi possível verificar-se os pedidos na *server_reservation_queue* a serem atendidos resultando
	Notas: Através deste teste foi possível validar as funcionalidades de criação de *reservation_data*, os mecanismos de proteção da corrupção de dados implementados com *mutex's* nas filas, a respetiva adição de entradas nas filas *server_reservation_queue* segundo o endereço MAC de destino da reserva, o atendimento dos comandos contidos na *server_reservation_queue* pela *thread tx_queue_handler* da interface correspondente, e as funcionalidades de *debug* criadas.	

Tabela A.6: Testes realizados para a validação do requisito Criação dos servidores fora das janelas temporais dedicadas para o envio de tráfego

Identificação do requisito de teste	***Descrição do cenário de teste***	***Descrição dos resultados obtidos***
Determinação do tempo de latência do switch	Foi concebido um cenário de teste onde se encontravam criados no sistema 25 reservas de recursos no sistema e por sua vez 26 servidores, incluindo o *background server*, numa específica interface do *switch* destinada à ligação do *Listener.* Com a utilização de uma estação terminal como *Talker* e uma outra como *Listener* ligada ao *switch* na interface antes referida, foi realizado através da ferramenta *PackEth* um fluxo de 50.000 pacotes com destino ao *Listener.* Foram colocados no sistema timers com o intuito de calcular o tempo utilizado pelo sistema para processar um pacote e o colocar no servidor menos prioritário correspondente ao 25º servidor acedido pelo módulo dos servidores planos.	Verificou-se por análise dos valores dos timers amostrados que o valor máximo que o sistema em *software* demorou a processar um pacote e a colocar no 25º servidor mais prioritário este foi de 4497ηs e o valor eficaz resultante do conjunto de amostras obtido foi de 1821,7519ηs.
	Notas: Para cálculos futuros foi utilizado o valor máximo obtido neste teste no algoritmo de decisão na variável referente ao tempo de latência do *switch.*	

Validação do algoritmo que rege o controlo de admissão	Foi concebido vários cenários de teste onde uma estação terminal funcionando como *Talker* vai realizar um pedido de reserva com determinadas características. Nos diferentes cenários de teste foram criados diferentes estados e características de reservas na *adv_SRDB* e por consequência também os servidores correspondentes a essas reservas. Os conjuntos de reservas no sistema criados posteriormente ao pedido de reserva por parte do *Talker* são constituídos por reservas com destino para o mesmo *Listener* decretado no pedido, assim como também para outros *Listeners* que não este. De tal forma o mesmo acontece para o campo designado ao endereço que identifica os *Talkers* da reserva. Desta forma foi concebido cinco conjuntos de reservas que garantissem matematicamente o cumprimento da *deadline*, aqui igual ao seu período, do envio dos pacotes segundo o algoritmo adotado para a admissão de pedidos de reserva e cinco em que não se consegue realizar tal garantia. Será analisado com recurso à consola correspondente do *switch* HaRTES os valores calculados para o tempo de resposta no *Uplink* e *Downlink* e por consequência o tempo de resposta total possível com os seguintes conjuntos.	Verificou-se que o algoritmo é capaz de calcular os tempos de bloqueio e de interferência consoante a origem e destino da *stream* para a qual se pretende realizar o pedido de reserva. Isto é possível através da análise dos endereços MAC de origem e destino presentes na tabela *adv_SRDB*. Desta forma verificou-se que o algoritmo no cálculo para o *Uplink* só tomou em conta os valores descritos nas reservas presentes no sistema provenientes do mesmo endereço MAC de origem que o do pedido de reserva a realizar. O mesmo se verificou para o cálculo no *Downlink*, em que o algoritmo só tomou em conta os valores descritos nas reservas presentes no sistema com o mesmo endereço MAC de destino que o do pedido de reserva a realizar. Finalmente, verificou-se que os tempos de resposta calculados pelo algoritmo implementado no sistema correspondem aos tempos esperados dos valores concebidos matematicamente, e que este é capaz de indicar se é possível no momento do cálculo fornecer a garantia de qualidade de serviço desejada por comparação do tempo de resposta total calculado com o valor da sua *deadline*.

	Notas: Para cálculos futuros com uma arquitetura de rede com *multi-hop* será necessário adaptar o algoritmo para juntamente com os dados armazenados na MAC *Address Fowarding Table* ser capaz de fazer a diferenciação das reservas a serem consideradas no *Uplink* e *Downlink* não pelos endereços MAC de origem e destino, mas pela interface de proveniência e saída de pacotes.

Tabela A.7: Testes realizados para a validação do requisito Mecanismo de controlo de admissão de pedidos de reserva

Identificação do requisito de teste	*Descrição do cenário de teste*	*Descrição dos resultados obtidos*
Validação do mecanismo de gestão da capacidade dos servidores	Foi concebido um cenário de teste onde duas estações terminais funcionam como *Talkers* e uma outra como *Listener.* No *switch* encontram-se realizadas duas reservas, entre cada um dos *Talkers* e o *Listener,* e por consequência dois servidores cada um criado para uma destas reservas. O servidor mais prioritário dos servidores criados apresenta uma capacidade nominal de 3036bytes e um período equivalente a dois EC's e o outro com uma capacidade nominal de 2000bytes e período equivalente a um EC. No *switch* o valor do EC encontra-se por defeito a 30000µs e o tamanho da *Ethernet Memory Pool* foi reduzido neste cenário de teste para o tamanho máximo de 20 pacotes de *Ethernet*. Através da ferramenta *PackEth* ambos os *Talkers* geram pacotes com um tamanho fixo de 1514bytes e com um intervalo entre envio de pacotes de 1000µs. Para monitorização do sistema foi utilizada a consola do processo correspondente ao *switch* HaRTES.	Verificou-se que o módulo dos servidores planos obriga a obedecer o tráfego definido pela capacidade e período para para cada servidor. À medida que os pacotes iam chegando estes foram sendo armazenados nos servidores respetivos das suas reservas acabando por os encher, no entanto a funcionalidade desenvolvida para *dispatch* de pacotes contidos nos servidores apenas permitiu o envio de dois pacotes a cada dois EC's ao servidor de maior prioridade, de um pacote por EC ao servidor menos prioritário.
	Notas: Através deste teste foi possível validar a gestão interna dos servidores planos, que neste teste consistia no mecanismo de refrescamento da capacidade instantânea dos servidores, a depleção desta capacidade à medida que os pacotes eram despachados, o mecanismo que garante que o uso de capacidade instantânea adicional à disponível não ocorre, e finalmente o despacho de pacotes segundo a prioridade e a capacidade disponível nos servidores. Adicionalmente também foi possível validar a funcionalidade de alocação de pacotes nestes servidores à medida que estes são processados pela *thread rx_multiplexer* e o descarte dos pacotes que não podem ser armazenados no sistema sem consequências na estabilidade do sistema.	

Tabela A.8: Testes realizados para a validação do requisito Despacho de tráfego assíncrono na janela correspondente e segundo um critério de prioridades e capacidade dos servidores